UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Predicción del número de individuos en una determinada localización

Alan Isachar Glotzer Barbuzano
Tutor: Rubén Hernando Martín
Ponente: Luis Fernando Lago Fernández

Mayo 2018

Predicción del número de individuos en una determinada localización

AUTOR: Alan Isachar Glotzer Barbuzano
TUTOR: Rubén Hernando Martín

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2018

Resumen

Este Trabajo de Fin de Grado ha sido desarrollado con el objetivo de construir un sistema que cuenta con una comunicación entre servicios fluida, alta escalabilidad y modularidad, y fácil integración, despliegue y adaptación. El proyecto es utilizado para predecir el número de individuos en una determinada localización y fecha futura, basándose en datos recogidos anteriormente. Por un lado, permite al usuario interactuar con una plataforma web en la que introduce su petición y, por otro lado, ésta se procesa utilizando un algoritmo desarrollado en Scala mediante Spark, comprimido en forma de JAR y actualizado automáticamente cada vez que se modifica gracias a un sistema de integración continua denominado Jenkins.

Para realizar las predicciones y desplegar los servicios de forma que sean accesibles para cualquier usuario, se ha hecho uso de servicios web de Amazon. Además, para la interacción con el usuario se ha utilizado Java para el back-end, junto con el framework Spring Boot, y HTML5, CSS, AJAX, JQuery y Javascript para el front-end.

El almacenamiento de las peticiones y de la información sobre los usuarios se realiza en una base de datos SQL, mientras que para almacenar los resultados calculados se ha hecho uso de una base de datos NoSQL, puntalmente MongoDB. El software desarrollado se integrará como un módulo en la plataforma de la empresa Infinia Mobile, el cual se utilizará para futuros análisis de selección de ubicaciones u horarios para desarrollar campañas publicitarias, asegurándose de que tengan el impacto deseado por los clientes de la empresa. La realización del proyecto ha seguido la metodología ágil Scrum con Sprints de tres semanas de duración, siendo un total de 6 Sprints.

Abstract

This Bachelor Thesis has been done to build a system with a fluid services communication, high scalability and modularity, and easy to integrate, deploy and adapt. This project is used to predict the number of people in a specific location and future date, based on previously collected data, as a proof of concept. On one hand, the system allows users to interact with a web platform, in which they can introduce a petition, and, on the other hand, it is processed through an algorithm made with Scala and Spark, compressed in a JAR and automatically updated when it changes, thanks to a continuous integration system called Jenkins.

In order to calculate predictions and deploy services in an accessible way, the project use Amazon web services. Furthermore, for user interaction it has been used Java for back-end development with Spring Boot framework, and HTML5, CSS, AJAX, Jquery and Javascript for front-end development.

The system uses an SQL database to store petitions and user information, while results are stored in MongoDB. Developed software will be integrated as an Infinia Mobile's platform module, which will be used for future advertising campaigns' deployments as a placement helper, in order to achieve Infinia Mobile's clients' desired impact. Project's development has followed an agile methodology called Scrum, with three weeks' Sprints, making a total of 6 Sprints.

Palabras clave (castellano)

HTML5, CSS, AJAX, JQuery, Amazon, Javascript, Mongo, Spark, Spring Boot, Back-end, Front-end, SQL, Jenkins, Docker, Redis, Gradle, S3, EC2, ECS, EMR, RabbitMQ, Scala.

Keywords (inglés)

HTML5, CSS, AJAX, JQuery, Amazon, Javascript, Mongo, Spark, Spring Boot, Back-end, Front-end, SQL, Jenkins, Docker, Redis, Gradle, S3, EC2, ECS, EMR, RabbitMQ, Scala.

Agradecimientos

Quiero agradecer a una cantidad importante de personas que me han apoyado y suministrado conocimientos desde el primer día hasta el último.

Por un lado, sin mi familia que me ha acompañado en todo momento animándome e impulsándome a conseguir mi objetivo no hubiera sido posible realizar este Trabajo de Fin de Grado. Por otro lado, no tengo palabras para agradecer lo mucho que me ha otorgado la empresa Infinia Mobile dándome la posibilidad de realizar este proyecto. He aprendido a manejarme con tecnologías que antes estaban totalmente fuera de mi alcance, y he aprendido a desarrollar un proyecto de principio a fin llevando una seriedad y rigor que en otras circunstancias no hubiera podido tener.

Dentro de la empresa, me gustaría agradecer en enorme medida a mi tutor Rubén Hernando, quien me ha otorgado la posibilidad de realizar este proyecto, suministrándome las herramientas necesarias, conocimientos y una guía y seguimiento a través de todo el desarrollo. Gracias a él este proyecto ha sido posible y se ha realizado de una manera ordenada, productiva y medida. Gracias por todo.

Además, agradecer especialmente a Víctor Cervera quién me ha instruido en las tecnologías y herramientas que yo desconocía, impulsándome y dándome soporte a lo largo de todo el proyecto. Desde un primer momento no tuvo ningún problema en echarme una mano, y en responder las dudas que me iban surgiendo a lo largo del desarrollo del Trabajo. Gracias por todo.

Infinia Mobile se compone de miembros serios, trabajadores, profesionales y con grandes conocimientos, encantados de echar una mano siempre que es necesario al compañero de la mesa de al lado. Dentro de esta sección, especial mención al CEO de Infinia Mobile, Fausto Fernández que, gracias a él, todo esto ha sido posible. También a Nicolás Diaz, quién aprobó la realización del proyecto y ayudó a la hora de decidir cómo realizarlo. Además, me gustaría hacer una mención especial a David Hernando, Jesús Calzada, Aday Robaina, Manuel Fernández, David Vega, Daniel Martínez y José María Viñes. Gracias por sacarme de un apuro más de una vez, y por suministrarme un ambiente de trabajo por el que estoy enormemente agradecido. Además del equipo técnico, hay tres miembros de la empresa que me ayudaron a dar mis primeros pasos dentro de Infinia Mobile y que, sin duda, han servido de apoyo a lo largo de todo el proyecto. Especial mención al equipo creativo: Rocío Diaz, Rafael García y Claudia Méndez. También a Tamara Cáceres por gestionar desde un principio mi incorporación y estancia en la empresa. Además de las menciones especiales, agradecer a todos los miembros de la empresa que, sin duda, han hecho de mi entorno laboral un sitio enormemente agradable.

Por último, agradecer a mi ponente Luis Lago, quién ha realizado un seguimiento del Trabajo de Fin de Grado desde principio a fin y que me apoyó durante el desarrollo de todo el proyecto suministrándome ideas, conceptos y características para la plataforma que han sido aplicadas en enorme medida.

GRACIAS

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

Desarrollar un proyecto completo con módulos independientemente actualizables que resulte en una plataforma web utilizada para predecir, con un pequeño margen de error y como prueba de concepto del sistema construido, cuántas personas habrá en una determinada localización en un determinada fecha, tras pasar grandes cantidades de datos por un algoritmo fácilmente adaptable y modificable, aporta a la empresa Infinia Mobile un módulo para que sus clientes se orienten a la hora de saber dónde invertir en publicidad para que tenga el efecto deseado.

La motivación por la que se desarrollará este proyecto es construir una plataforma con la que el usuario interactúe y sea capaz de solicitar predicciones, ver las anteriormente realizadas y sus resultados. Al estar realizado en módulos separados, su integración con otras plataformas es sencilla y permite que se modifique cada apartado por separado, como si fueran proyectos independientes.

1.2 Objetivos

El objetivo de este Trabajo de Fin de Grado es diseñar, definir, implementar y validar un módulo completamente operativo capaz de predecir la cantidad de personas en una determinada fecha y localización, así como de interactuar con el usuario para transmitir toda la información. Como objetivo adicional, el módulo será desarrollado para una integración con la plataforma de la empresa Infinia Mobile.

El resultado que se obtiene de este trabajo es, por tanto, la construcción de *Prediction*, módulo de predicción de datos. Para poder llevar a cabo todas sus funcionalidades se necesita:

- Un fácilmente modificable algoritmo de predicción
- Un proceso de Big Data que ejecuta el algoritmo desarrollado
- Una intuitiva interfaz gráfica para el usuario que solicita los datos
- Una gestión de usuarios en la plataforma
- La transmisión, resolución y visualización de resultados de las peticiones
- Un acceso a dos bases de datos distintas
    - MongoDB para almacenar los resultados
    - SQL para almacenar los usuarios y peticiones
- Una gestión del software de integración continua
- Un despliegue del software en los servicios web de Amazon
- La configuración de las máquinas que resuelven las peticiones
- La comunicación entre los distintos servicios desarrollados

1.3 Introducción a Infinia Mobile

Para poner en contexto el desarrollo del trabajo es necesario introducir la empresa para la que se realiza. Infinia Mobile cuenta con una plataforma de gestión de datos (DMP – Data Management Platform) que genera clusters de audiencia en función de sus hábitos de vida, con un objetivo de crear un DMP con datos propios capaces de extraer comportamientos de los usuarios.

Para interactuar con los distintos clientes cuenta con una plataforma que otorga la posibilidad de gestionar las campañas publicitarias de forma rápida e intuitiva. Durante las prácticas en empresa colaboré en el desarrollo de esta y ahora, en el Trabajo de Fin de Grado, desarrollo un módulo predictivo para esta plataforma.

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1: Introducción
- Capítulo 2: Estado del arte
- Capítulo 3: Análisis y diseño
- Capítulo 4: Desarrollo
- Capítulo 5: Integración, pruebas y resultados
- Capítulo 6: Conclusiones y trabajo futuro

2 Estado del arte

2.1 Introducción

Tradicionalmente las empresas que desean mostrar publicidad lo hacen de forma genérica y sin información con respecto a los usuarios puntuales a los que influye. Esto ha ido evolucionando con el tiempo, actualmente la personalización de los anuncios mostrados se realiza de una forma eficaz. En el caso de Infinia Mobile, cuenta con una SDK (Software Development Kit) que se integra en distintas aplicaciones de iOS y Android que ingesta datos del usuario de ese dispositivo como información de este, aplicaciones instaladas o localizaciones por las que pasa. Todos los datos recogidos desde la SDK nos permiten perfilar a los usuarios acorde a sus gustos o intereses.

Para el caso puntual de este trabajo, la información necesaria para las predicciones es: localización (latitud, longitud) y fecha en la que ha sido recogida. Ésta se ha obtenido gracias a la SDK instalada en aplicaciones y se ha almacenado en ficheros que residen en servidores de Amazon (S3) para realizar el análisis y predicciones.

Para poder aprovechar esta información recogida de una nueva manera, se ha desarrollado una plataforma utilizando una gran cantidad de herramientas y tecnologías. A la hora de desarrollarla, se ha realizado un estudio acerca de cuáles son las más utilizadas actualmente, y qué ventajas proporciona cada una.

2.2 Tecnologías existentes

Este Trabajo de Fin de Grado ha utilizado una gran cantidad de herramientas que serán explicadas a continuación, así como se expondrán otras posibles alternativas. Para más claridad a la hora de la lectura, se ha dividido en 5 partes el desarrollo del proyecto:

- Realización la plataforma web:

  Para llevarla a cabo se ha utilizado el framework **Spring Boot** [1], que permite crear la aplicación y desplegarla de una forma sencilla, así como realizar una configuración avanzada con módulos como **Spring Security**. Éste provee una autenticación y autorización de aplicaciones Java, aplicado en el caso de este proyecto para la realización del login, registro y gestión de permisos. Para obtener de forma gráfica la información relativa al despliegue de la aplicación, se recomienda acceder al apartado E de la sección anexos de la memoria, donde se encuentra un diagrama de despliegue. Una alternativa a Spring Boot podría haber sido Dropwizard, herramienta que provee características similares. La elección de una frente a otra fue hecha para facilitar la integración con la plataforma de Infinia Mobile realizada con Spring Boot.

Figura Estado del Arte-1: Logo de Spring Boot

A la hora de mantener las sesiones de los usuarios activas, se ha recurrido a la utilización de **Redis 4.0.2** [2], motor de base de datos en memoria. Para la parte de front-end, se han utilizado lenguajes de programación tales como **HTML**, en conjunto con **CSS**, **Javascript**, **AJAX** y **Jquery**. En el caso particular de este proyecto, Javascript ha sido de gran utilidad para realizar las peticiones a la API del back-end para recuperar los datos necesarios o para enviar las peticiones de predicción de los usuarios. También ha sido utilizado para representar los mapas de Google Maps gracias a su API. Para poder seguir utilizando la plataforma mientras se realizan las predicciones, se ha hecho uso del antes mencionado AJAX.

Figura Estado del Arte-2: Herramientas front-end

Para automatizar la construcción de código, se ha utilizado la herramienta **Gradle 4.2.1**. [3] Éste trabaja sobre Maven y Ant, siendo una herramienta software que gestiona y construye proyectos Java, con funcionalidades similares a las de Ant, salvando las diferencias, ya que éste tiene un simple modelo de configuración basado en XML (Extensible Markup Language).

- Realización del algoritmo

Para realizar el algoritmo se ha utilizado **SparkSQL 2.2.1**. [4] Spark es un framework de computación en cluster que permite trabajar con paralelismo de datos de forma veloz y óptima en recursos. En concreto, SparkSQL permite el procesamiento de datos estructurados basados en SQL. Éste tiene dos tipos de operaciones básicas posibles a ser ejecutadas sobre una RDD (Resilient Distributed Dataset): transformaciones y acciones. Las primeras cambian el flujo de información generando una nueva RDD, mientras que las segundas evalúan dichos dataset y devuelven resultados.

Scala es el lenguaje utilizado junto con SparkSQL para desarrollar el algoritmo de procesamiento de datos que contiene el JAR. Este lenguaje multi-paradigma ha sido elegido ya que permite el desarrollo de aplicaciones Spark de forma sencilla y eficiente en cuanto a operaciones y funciones. También se podría haber utilizado Python, pero se ha elegido Scala ya que es funcional de forma nativa.

Figura Estado del Arte-3: Herramientas para la realización del algoritmo

- Servicios Web de Amazon

  A la hora de desplegar los servicios y procesar las peticiones, se ha hecho uso de Amazon Web Services. En concreto, **Amazon S3** [5] se ha utilizado para almacenar los datos en los que se basan las predicciones. A su vez, contiene también el JAR que procesa el algoritmo de predicción. Por otro lado, **Amazon EC2** [6] posee la configuración y gestión de las instancias, grupos de seguridad, IP elásticas y balanceadores de carga de la aplicación. Además, se ha hecho uso de **Amazon ECS** [7]**,** servicio de organización de contenedores de alta escalabilidad, compatible con Docker, que le permite ejecutar aplicaciones en contenedores con facilidad.

  Por último, **Amazon EMR 5.11.0** [8] se ha utilizado como gestor de instancias Amazon EC2 que proporciona un marco Hadoop para procesar grandes cantidades de datos. EC2 permite alquilar ordenadores virtuales donde ejecutar aplicaciones propias y Hadoop es un framework que soporta aplicaciones distribuidas, es decir con distintos componentes que se ejecutan en entornos separados. En EMR se levantan los clusters que procesan las peticiones pedidas por los usuarios.

Figura Estado del Arte-4: Amazon Web Services

- Comunicación entre servicios

  Como software de intercambio de mensajes se ha utilizad **RabbitMQ 3.5.7**. [9] Más precisamente, se ha utilizado para la comunicación entre servicios del back-end de la aplicación, así como para comunicar estos con el JAR encargado del algoritmo de procesamiento de datos.

  Para que los usuarios puedan acceder a través de internet a la plataforma web, se ha hecho uso de **Docker 17.11.0** [10] en conjunto con Amazon ECS. Las imágenes de los servicios back-end realizados con **Java,** y lenguajes de programación web mencionados anteriormente como HTML, se crean gracias a Docker y se suben a ECS para ser desplegadas como servicios.

  Como sistema de integración continua se utiliza **Jenkins** [11]. Gracias a esta herramienta cada vez que se detecta un cambio en el repositorio donde está almacenado todo el código, se ejecutan automáticamente los tests y se actualiza el JAR con el algoritmo de procesamiento de datos.

Figura Estado del Arte-5: Herramientas para la comunicación entre servicios

- Herramientas adicionales e IDE (Integrated Development Environment)

  Como IDE principal se ha utilizado **Eclipse**, plataforma de software compuesto por herramientas de programación. Este ha sido utilizado para facilitar la implementación de la aplicación web, tanto la parte de front-end, como la de back-end y su integración. Permite gestionar visualmente los distintos servicios como proyectos independientes que se comunican entre ellos.

  Para desarrollar el algoritmo de procesamiento de datos en Scala y el JAR que lo contiene, se ha trabajado en **IntelliJ**. Tras codificar el algoritmo se utiliza un empaquetado para convertirlo en JAR y ser ejecutado en los servicios de Amazon.

  Como herramienta adicional, **XAMPP**, servidor web de plataforma que permite la gestión de bases de datos SQL, utilizar servidores web Apache e intérpretes de PHP y Perl, se ha utilizado en este proyecto para permitir el almacenamiento de datos en bases SQL en las pruebas en local.

  RoboMongo (Robo 3T – 1.1) es un GUI (Graphical User Interface) que se ha utilizado para gestionar la base de datos de MongoDB, permitiendo las conexiones a la misma de forma sencilla, así como la visualización y edición de sus colecciones.

  Otro GUI utilizado ha sido **MySQL Workbench**. En este caso se ha utilizado para gestionar las bases de datos SQL, permitiendo una sencilla visualización, edición y gestión de las tablas y relaciones entre las mismas.

  Además, **PostMan** (ADE - API Development Environment) se utilizado para las pruebas de correcto funcionamiento de la interfaz de programación de aplicaciones desarrollada.

  Por último, **Ganglia 3.7.2** herramienta de monitoreo de forma distribuida y escalable para sistemas de cómputo de alto rendimiento, clusters y redes, ha servido para evaluar la gestión de recursos de los clusters que realizan las peticiones de predicción.

Figura Estado del Arte-6: Herramientas adicionales e IDE

En cuanto a la parte del algoritmo de predicción y para comprar el trabajo que desarrollado con empresas que actualmente llevan unas tareas similares, se ha hecho un estudio de empresas que utilizan la agrupación de técnicas llamada Análisis Predictivo. Éstas son técnicas estadísticas de modelización y minería de datos que analiza información para hacer predicciones del futuro.

Actualmente hay bastantes empresas que utilizan estas técnicas, algunas de ellas son las presentadas a continuación:

IBM

IBM [12] cuenta con un módulo dentro de su empresa llamado IBM Analytics. Éste utiliza algunas de las mismas herramientas utilizadas en este proyecto, como pueden ser Hadoop y Spark.

Figura Estado del Arte-7: Logo de IBM

Ofrecen gran cantidad de funcionalidades, como pueden ser servicios de datos en la nube o gestión de datos y contenido. Estos aspectos en mi proyecto están gestionados en servidores de Amazon. La más relevante para este estudio de tecnologías existentes trata acerca de la analítica predictiva para BigData. Útil recurso para empresas que almacenan gran cantidad de datos y deseen obtener relevante información de estos.

En comparativa, este servicio realiza una tarea similar a mi Trabajo de Fin de Grado, siendo la principal diferencia que en este caso se basa únicamente en predicción mediante localizaciones.

SAS

Esta empresa se especializa en Analítica de Software y en aportar soluciones a sus clientes. Concisamente, el departamento que se ocupa de la predicción de datos se llama SAS [13] Visual Statistics.

Figura Estado del Arte-8: Logo de SAS

Éste último otorga la posibilidad de explorar datos, crear y definir modelos predictivos y aumentar la productividad analítica. Una de las características que promueven en gran medida es la visualización gráfica de los datos, lo cual facilita la interacción con el cliente.

Para el seccionamiento de datos utilizan Clustering como es nuestro caso, lo cual otorga paralelismo y escalabilidad en las operaciones. Por otro lado, su algoritmo de predicción trabaja con árboles de decisión, donde el cliente tiene la posibilidad de interactuar con los mismos de forma gráfica. Por otro lado, trabajan con regresión logística y lineal.

Otorgan por otro lado una visión estadística de los resultados, donde utilizan histogramas y tablas personalizables para facilitar el entendimiento a los clientes. De esta empresa se puede aplicar a mi Trabajo de Fin de Grado la potente interfaz gráfica

que poseen de representación de datos. Cuanta más información se consiga de los mismos, se está realizando un mejor aprovechamiento del análisis.

SAP

Esta empresa tiene un importante lugar en el mercado de procesamiento de datos. Cuenta con el producto SAP [15] HANA que otorga la posibilidad de tratar grandes volúmenes de datos en tiempo real sin latencia. Cuenta con uno de los mayores almacenes de datos actualmente registrados.

Figura Estado del Arte-9: Logo de SAP

Las principales ventajas que aporta HANA son claves dentro del análisis de datos. Por un lado, cuentan con una herramienta que resume en una sola plataforma toda la información relevante para el cliente. Esta hace uso de tecnologías similares a la de mi Trabajo de Fin de Grado, como pueden ser Hadoop, bases de datos NoSQL o Spark.

El tratamiento de datos en tiempo real capaz de ser desplegado en cualquier entorno, tanto privado (local) como público (en la nube) otorgan una ventaja de gran importancia a la plataforma. El análisis de datos puede ser realizado mediante distintas técnicas: virtualización de datos, la posibilidad de manipular datos sin requerir su información técnica; integración de datos, combinando información que reside en diferentes recursos y réplica de datos: mejorando así la confiabilidad, la tolerancia al fallo y la accesibilidad.

Otorgar este elevado nivel de análisis de datos a este Trabajo de Fin de Grado es un objetivo que se podría tener en cuenta en futuras implementaciones que, sin duda, se basarían en gran medida en HANA.

Oracle

La conocida empresa Oracle [15] que presta servicios de soluciones de nube y locales cuenta con un módulo llamado Oracle Advanced Analytics que integra Oracle R Enterprise y Oracle Data Mining.

Figura Estado del Arte-10: Logo de Oracle

El primero se centra en el análisis de datos en cualquier base de datos, mientras que el segundo trata modelos de análisis predictivos.

Los beneficios que aporta pueden ser resumidos en principalmente tres. Por un lado, cuenta con un nivel de escalabilidad y rendimiento elevado, un modelo avanzado de predicción de datos donde el procesamiento y modelado de los mismos ocurre en un periodo corto de tiempo, otorgando al cliente resultados casi inmediatos y por último cuenta con servidores de análisis integrados, por lo que no es necesario contratarlos de manera externa.

Para llevar a cabo el procesamiento de datos trabaja con herramientas como SQL, APIs de R o Clusters. Dependiendo de cada cliente, el modelo de trabajo que ofrecen se adapta al coste y capacidad que cada uno necesita, siendo flexible en este aspecto. Además, otorgan un sistema de alta escalabilidad y seguridad.

Microsoft

Mediante el módulo SQL Server de Microsoft [16] ofrecen un análisis de datos rápido, flexible, seguro y escalable, características indispensables en este tipo de tareas. Ofrecen un servicio de análisis de negocio mediante Data Mining Add-ins, que permite extraer patrones y visualizarlos de manera
sencilla a través de una interfaz gráfica.

Figura Estado del Arte-11: Logo de Microsoft

Por otro lado, esta empresa hace uso de SQL Server Management Studio, una aplicación que aúna toda la configuración, administración y tratamiento de los distintos componentes del módulo SQL Server.

El valor de los datos que las empresas recogen reside en la información que son capaces de extraer de los mismos. Microsoft otorga la posibilidad de analizar y utilizar datos que, en otras ocasiones, se hubieran desaprovechado mediante herramientas ya mencionadas como Hadoop o Spark. Para almacenar y tratar los mismos, cuenta con una herramienta llamada Azure Data Lake, repositorio de gran escala que carga con el análisis de datos.

2.3 Conclusiones

Tras el estudio de tecnologías de implementación de plataformas web, despliegue de aplicaciones y gestión de bases de datos, así como de empresas existentes que se dedican al análisis y predicción de datos, he podido obtener referencias acerca de cómo desarrollar un sistema que cumpla las necesidades del cliente y se adapte a la tecnología actual.

Como características relevantes que debe poseer mi Trabajo de Fin de Grado podemos encontrar la facilidad de uso para el cliente, la escalabilidad, representación visual de datos y resultados, posibilidad de modificar cada uno de los módulos por separado sin que el resto se vean afectados y una fácil integración con otras plataformas.

La necesidad de la que surge este proyecto es concisa, desarrollar una plataforma operativa con bases de datos, despliegues, uso de herramientas actualmente cumbres en la tecnología software, aplicada en este caso a la predicción de localizaciones mediante datos recopilados anteriormente. Las ventajas que aporta el sistema desarrollado son su escalabilidad, modularidad y fácil integración, despliegue y adaptación.

3 Análisis y Diseño

3.1 Análisis

3.1.1 Requisitos funcionales y no funcionales

A continuación se encuentran los requisitos con los que debe contar el sistema para cubrir todos los objetivos:

3.1.1.1 Requisitos funcionales

RF. 1 La aplicación permitirá a los usuarios introducir sus datos para entrar en el sistema

RF. 2 El sistema dará la opción de elegir una localización, radio y fecha determinados para obtener una estimación del número de personas que habrá bajo esos parámetros

RF. 3 El sistema notificará al usuario que su petición se está procesando

RF. 4 El sistema permitirá al usuario ver las peticiones que ha realizado

RF. 5 El sistema permitirá al usuario ver los resultados obtenidos de sus peticiones

RF. 6 El sistema admitirá dos roles para los usuarios, "Admin" y "User"

RF. 7 Un administrador podrá registrar nuevos usuarios en la aplicación

RF. 8 Los usuarios podrán cerrar sesión cuando deseen, abandonado el sistema

RF. 9 La aplicación será gestionada por un software de integración continua

3.1.1.2 Requisitos no funcionales

RNF. 1 La aplicación tendrá una interfaz gráfica sencilla e intuitiva para el correcto uso de los usuarios

RNF. 2 Se podrá acceder a la aplicación desde cualquier navegador

RNF. 3 La aplicación permitirá continuar con su uso mientras realiza los cálculos predictivos

RNF. 4 El sistema será realizado de forma modular para facilitar la modificación individual de cada parte

RNF. 5 La capacidad de procesamiento del sistema podrá escalar fácilmente si en un futuro fuera necesario

RNF. 6 El sistema será accesible desde una URL desplegada en Amazon Web Services

RNF. 7 El sistema trabajará con dos bases de datos, una SQL para usuarios y peticiones y una en MongoDB para almacenar los resultados

RNF. 8 Las imágenes de los servicios serán generadas con Docker y se almacenarán en repositorios de Amazon Web Services

RNF. 9 Los datos utilizados para calcular las predicciones estarán almacenados en Amazon S3

3.1.2 Funcionalidades principales del sistema

El sistema va a comunicarse con el usuario a través de una plataforma web o directamente mediante peticiones a la API. Cuando el usuario introduzca sus credenciales, podrá realizar peticiones de predicción, visualizar sus peticiones, resultados obtenidos y, por último, cerrar sesión.

Para que los usuarios puedan interactuar con el sistema, este será desplegado en servicios web Amazon. Las peticiones se resolverán en los mismos y, tras ello, se almacenarán en una base de datos Mongo. Los usuarios y las peticiones se almacenarán en una base de datos SQL, donde se indicarán los estados de las peticiones, mostrando los resultados en la aplicación web de las ya procesadas.

La comunicación entre servicios se realizará mediante una cola de mensajes. La aplicación estará cubierta por un sistema de integración continua, el cual actualizará automáticamente el algoritmo cuando éste se modifique y ejecutará los tests de la aplicación cuando haya un cambio en esta.

3.2 Diseño

3.2.1 Arquitectura general

Este Trabajo de Fin de Grado está desarrollado en base a una arquitectura que une front-end, back-end, bases de datos y procesamiento de datos. Dicha estructura se define de la siguiente manera:

- Mediante un front-end desarrollado para poder ser conectado de forma sencilla con el back-end, el usuario interactúa con la aplicación: login, petición de resultados, visualización de peticiones, visualización de resultados, etc. Este módulo del proyecto ha sido desarrollado en HTML5, Javascript, CSS, AJAX y JQuery y se comunica con el back-end mediante peticiones HTTP.

- Utilizando la herramienta Spring Boot integramos el módulo anteriormente mencionado con el back-end. Éste ha sido desarrollado en Java y es quien procesa las peticiones que el usuario realiza. Cuenta con la implementación de colas RabbitMQ para realizar trabajos de manera asíncrona sin interrumpir el flujo de la aplicación web. En este caso son utilizadas para comunicarse con el proceso que utiliza el JAR encargado de calcular los resultados mediante la aplicación de un algoritmo desarrollado.

- Dicho algoritmo ha sido implementado en Scala utilizando Spark. Tras su compilación se realiza un empaquetado en forma de Uber JAR (JAR con todas

sus dependencias incluidas) que se almacena en el servicio S3 de Amazon para que pueda ser accedido por los servidores del cluster a la hora de la ejecución.

- El sistema de clusters está almacenado por Amazon EMR, que se encuentra a su vez conectado con el back-end del sistema. Este servicio está configurado para procesar las peticiones en steps, que son agregados a clusters. Si ya existe uno creado procesando, se le añaden al mismo. Si no, se levanta uno nuevo. Para no incurrir en gastos innecesarios, cuando un cluster termina todos los steps que tiene asignados, termina de forma automática. De esta manera se asegura tener capacidad para procesar todas las peticiones, sin utilizar recursos innecesarios.
  A continuación, se presenta un diagrama de la arquitectura general de la aplicación, para que pueda ser visualizada con facilidad:

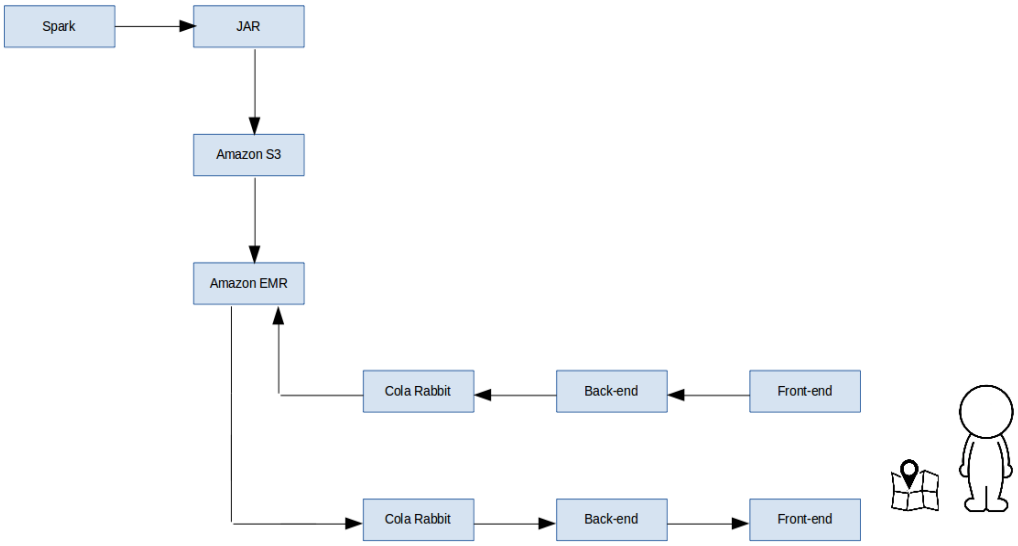


Figura Estado del Arte-12:Esquema de la arquitectura general

3.2.2 Interfaz de usuario

La interfaz de usuario es la parte realizada como front-end de la aplicación. En este, el usuario tiene varias acciones a realizar:

- Login: como toda aplicación, el usuario es capaz de introducir sus credenciales para acceder a la plataforma.

Figura Diseño-13: Pantalla de login

- Petición de resultado: para realizar una predicción, el usuario introduce una latitud, longitud, radio y fecha deseada. Tras esto, el sistema le muestra un aviso de que su petición se está procesando y el usuario puede seguir navegando por la plataforma. Cuando el resultado esté disponible podrá visualizarlo como se explica en el siguiente punto.

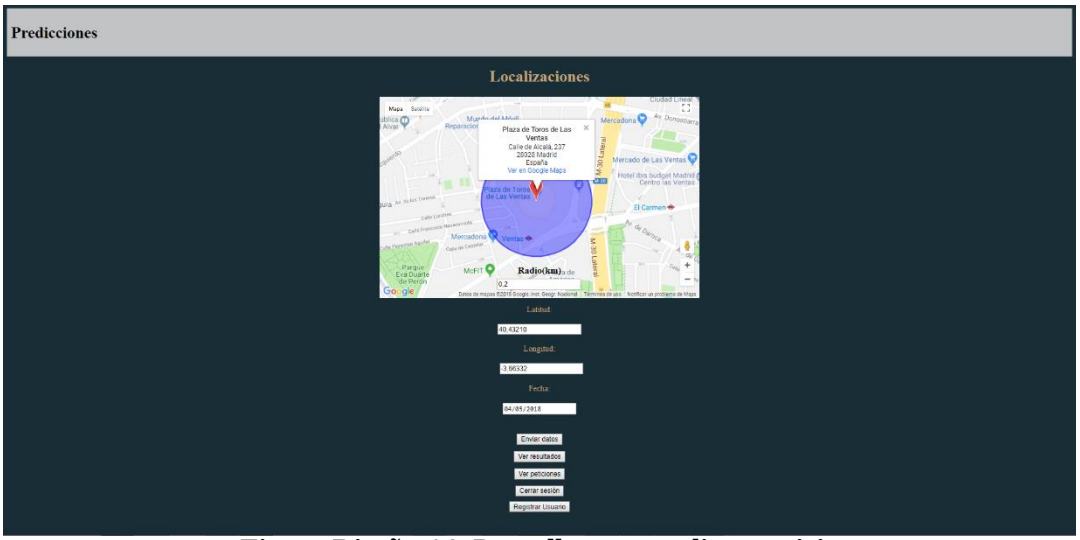


Figura Diseño-14: Pantalla para realizar peticiones

- Visualización de resultados: el usuario puede solicitar visualizar todos los resultados de sus peticiones una vez estén procesadas.

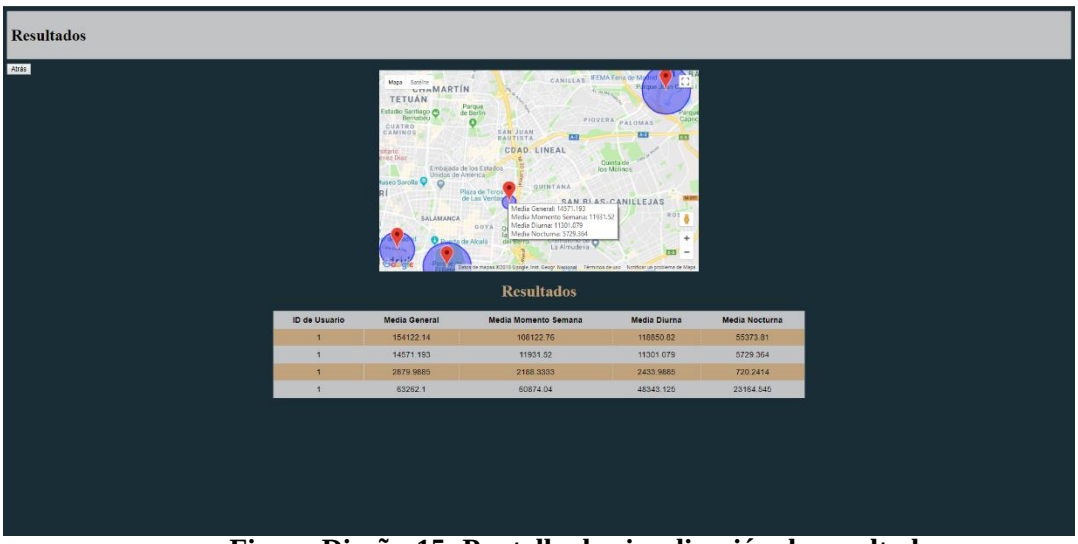


Figura Diseño-15: Pantalla de visualización de resultados

- Visualización del historial de peticiones: para que el usuario sepa si su petición se ha terminado de procesar o no, siempre que quiera puede acceder a esta información en este apartado, ya que se muestra el estado de cada una.

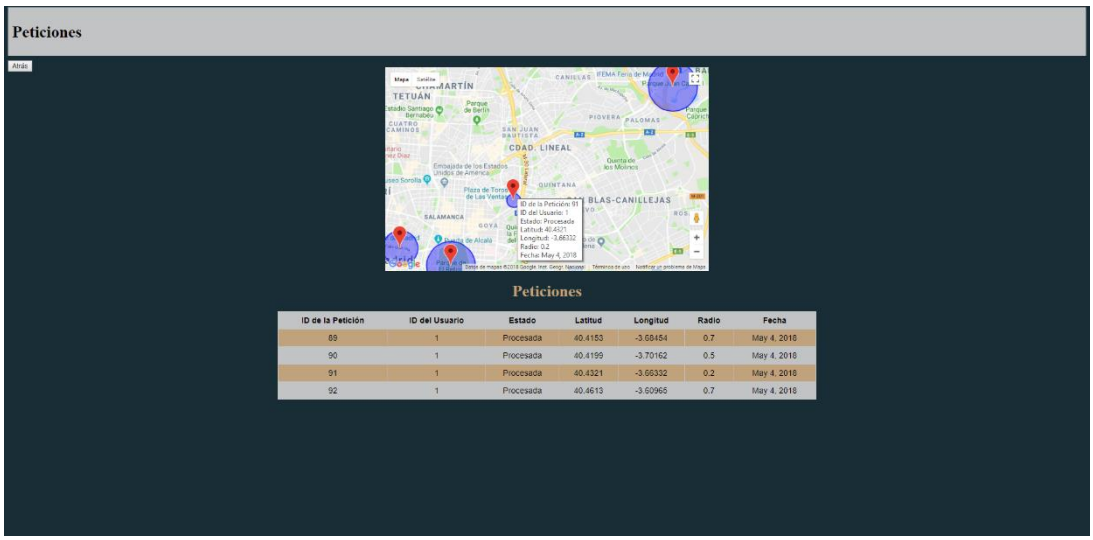


Figura Diseño-16: Pantalla de visualización del historial de peticiones

- Registro de usuario: cuando el usuario activo tiene rol de administrador, éste puede crear nuevos usuarios y elegir sus roles.

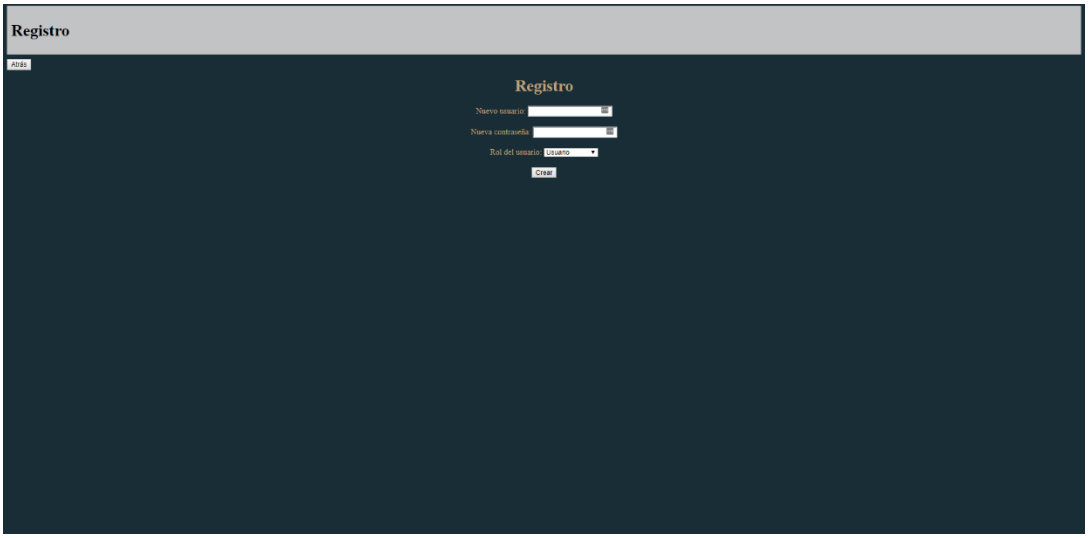


Figura Diseño-17: Pantalla de registro de usuario

3.2.3 Bases de datos

El sistema cuenta con dos bases de datos, una para almacenar las peticiones, roles, usuarios y relaciones usuarios-roles (SQL) y otra para almacenar los resultados (MongoDB).

- A la hora de tener información acerca de las peticiones de usuario, roles y usuarios, es de especial utilidad tener acceso a una base de datos relacional, ya que la conexión entre tablas como usuario y peticiones simplifican mucho las consultas. La base de datos cuenta con las siguientes tablas, cuyas relaciones se explican en la figura 18:

  - Location (peticiones en ciertas localizaciones):
    - ID: Identificador de la petición.
    - Status: Las peticiones tienen 3 estados. “Pendientes” (2): Peticiones que se están procesando en Amazon EMR, pero para las cuales aún no se han obtenido resultados. “Finalizadas” (3): Peticiones cuyos resultados están disponibles para el usuario. “Indefinidas” (Cualquier otro valor): Peticiones en las que se ha encontrado un error al ser procesadas.
    - Date: Fecha (timestamp) en la que predecir resultados.
    - Latitude: Latitud introducida por el usuario para realizar la predicción.
    - Longitude: Longitud introducida por el usuario para realizar la predicción.
    - Radio: Radio introducido por el usuario para realizar la predicción.
    - User_ID: Identificador del usuario que realiza la petición.
  - Role (roles de los usuarios):
    - ID: Identificador del rol.
    - Status: Estado del rol.
    - Role: Nombre del rol. La aplicación cuenta con administradores (Admin) y usuarios (User). Los Admin pueden crear nuevos usuarios y designarles

un rol, y ver las peticiones y resultados de todos los usuarios, mientras que los User solo pueden ver las suyas.

- User (usuarios de la aplicación):
  - ID: Identificador del usuario.
  - Status: Estado del usuario.
  - Active: 1 – Usuario activo, 0 – Usuario inactivo.
  - Password: Contraseña del usuario.
  - Username: Nombre de usuario de la contraseña.
- User_roles (asociación usuario-rol)
  - Userdao_id: Identificador del usuario.
  - Roles_id: Identificador del rol.

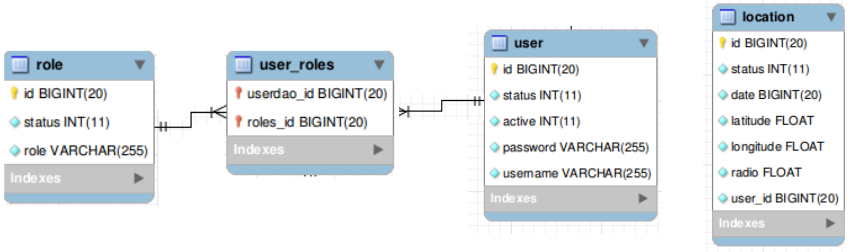


Figura Diseño-18: Diagrama relacional de la base de datos SQL

- Los resultados deben almacenarse en una base de datos no relacional como MongoDB, ya que en caso de modificar el algoritmo, los resultados almacenados no tienen por qué ser los mismos, y es necesaria una flexibilidad en este aspecto.

  Campos de la base de datos con el algoritmo actual:

  - _id: Identificador del objeto.
  - average: Resultado de la media calculada.
  - averageWeek: Resultado de la media en función de si es día de semana o fin de semana.
  - averageDay: Resultado de la media en horario laboral.
  - averageNight: Resultado de la media en horario de descanso.
  - userId: Identificador del usuario para el que se han obtenido los resultados.
  - lat: Latitud para la que se han obtenido los resultados.
  - lon: Longitud para la que se han obtenido los resultados.
  - radio: Radio para el que se han obtenido los resultados.

  Los cálculos de los distintos resultados se explican en el apartado 4.2.7 de la memoria.

4 Desarrollo

4.1 Introducción

El desarrollo de este Trabajo de Fin de Grado abarca distintas herramientas y tecnologías que se explicarán a continuación, así como su utilización e integración.

4.2 Estructura del sistema

4.2.1 Plataforma Web

Para que el usuario interactúe con las predicciones de datos, se ha desarrollado una plataforma utilizando la herramienta Spring Boot. Todas las clases necesarias para el funcionamiento de la plataforma web han sido representadas mediante un diagrama de clases en la sección C de los anexos.

Spring es un framework utilizado para el desarrollo de aplicaciones y contenedores. Tiene diversas herramientas como Spring Data o Spring Security, las cuales se pueden configurar de manera individual o utilizar Spring Boot, el cual sincroniza las distintas tecnologías y permite manejarlas todas en conjunto. Para identificar las entidades con las que trabaja hace uso de distintas etiquetas tales como Service o Repository, que indican de qué tipo de entidad se trata cada una.

En concreto, Spring Security ha sido la herramienta utilizada para configurar el login y registro de los usuarios. Mediante la clase WebSecurityConfig se ha configurado la autenticación y se han establecido los endpoints de la aplicación a los que se tiene acceso estando sin autenticar, y para los que es necesario una autenticación. Los usuarios se almacenan en la base de datos SQL al ser registrados y al hacer login se recurre a esa información.

Para desarrollar e integrar distintos proyectos, Spring Boot utiliza Maven o Gradle. En este caso se utiliza la segunda opción, la cual es un sistema de automatización de construcción de código abierto que a su vez construye sobre Maven para gestionar la construcción. Éste utiliza un POM (Project Object Model) [17] para describir el proyecto de software.

El proyecto está desarrollado en base a microservicios. En este caso, cuenta con 4 servicios realizados sobre Spring Boot. Estos son discovery-service, front-server, gateway-service y prediction-service:

- Discovery-service: este servicio se encarga de registrar al resto para que puedan llevar a cabo su comunicación. Cada servicio, incluido el discovery, se levanta en un puerto distinto y se registra. Spring trabaja con una herramienta llamada Eureka, que cuenta con una interfaz gráfica de visualización de servicios registrados:

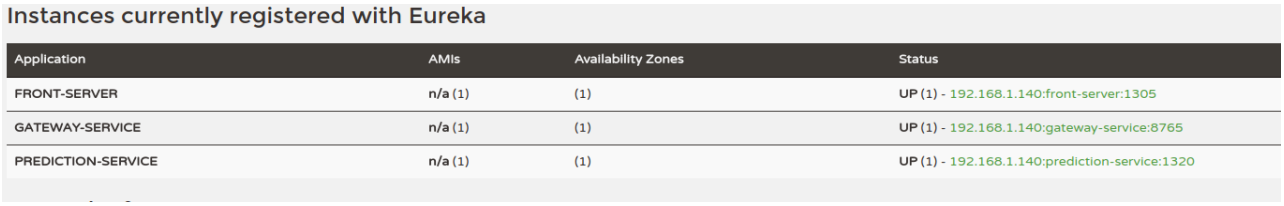


Figura Desarrollo-19: Instancias registradas en Eureka

- Front-server: se encarga de gestionar la interfaz gráfica de la aplicación. Por un lado, como la mayoría de los servicios, tiene un controlador que gestiona las peticiones que el usuario realiza. A diferencia del resto, éste no se trata de un RestController donde se devuelve el objeto pedido y sus datos escritos directamente en la respuesta HTTP como un JSON o XML, si no que se trata de un Controller que trabaja con la tecnología View que permite mostrar templates desarrollados en la aplicación.

  Éstos se encuentran en una carpeta distinta al código Java, resources/templates. Ésta ruta es necesaria para que Spring detecte las vistas. Para entender cómo está distribuida la interfaz gráfica de la aplicación, a continuación, se explica cada una de las vistas desarrolladas en HTML con recursos JavaScript, AJAX y Jquery:

  - login.html: el usuario debe introducir sus credenciales para poder continuar con la aplicación. Gracias a la herramienta Spring Security, el sistema no deja acceder a otras partes de la aplicación ni realizar peticiones a menos que el usuario se haya registrado.

  - dashboard.html: en esta pantalla el usuario tiene la posibilidad de realizar las peticiones. Para ello se le muestra un mapa gracias a la Tecnología de Google Maps, donde puede indicar una ubicación y un radio. La vista traduce esta información a los campos de un formulario en el que se envía la información. También se puede indicar de forma manual qué latitud, longitud se desea, así como la fecha de la predicción. También es posible navegar a la pantalla de visualización de resultados y peticiones.

  - peticiones.html: esta vista se encarga de mostrar las peticiones que el usuario ha realizado. Para ello, hace una petición POST al endpoint correspondiente a las peticiones (más adelante se explicará su funcionamiento con los controladores), se procesan los datos devueltos y se añaden a la tabla. También se traducen los estados almacenados en la base de datos en forma numérica a texto, para que el usuario lo entienda correctamente (petición procesada, finalizada o estado indefinido).

  - resultados.html: esta vista tiene un comportamiento similar a la de peticiones, hace una petición POST a su endpoint correspondiente, donde obtiene los datos de los resultados pedidos, y los pinta gráficamente en una tabla que se muestra por pantalla.

  Por último, este servicio está protegido ante usuarios sin autentificar mediante la implementación de WebSecurity, quien bloquea las peticiones tanto GET como POST a dichos usuarios.

- Gateway-service: los servicios que se registran en Discovery-service son accedidos por el usuario a través de un Gateway, quien redirige las peticiones según corresponde; no se accede directamente a ningún servicio.

  Por otro lado, este servicio se encarga de gestionar la parte de seguridad del login, redirigiendo al usuario a la página principal tras autenticarse de manera correcta o, por el contrario, informándole que ha introducido de forma errónea sus credenciales.

- Prediction-service: este servicio es el encargado de procesar las peticiones que se realicen al back-end, tras ser enrutadas a través del gateway. Es quien gestiona los endpoints de la API del sistema.

  Para evitar las peticiones sin autenticar del usuario, cuenta con Spring Security que bloquea las llamadas GET y POST si es necesario. Además, cuenta con los controladores de las llamadas a la API. Primero, un controlador genérico cuenta con endpoints que permiten añadir datos a un servicio, devolver todos los datos almacenados o uno específico filtrando por ID. De este controlador (GenericController) extiende el resto.

  Los usuarios tienen un controlador específico que gestiona sus peticiones (UserController). El primer endpoint destacable a mencionar es el que permite la comunicación vía colas RabbitMQ con los servicios específicos de dentro de Prediction-service. Éste se encarga de recibir la información del front-end cuando el usuario realiza una petición, almacenarla en la base de datos y notificar al servicio QueueManager, quien se encarga de la comunicación con Amazon EMR y Amazon S3, que será explicada más adelante.

  Otra funcionalidad importante de la que se encarga es de gestionar las peticiones de obtención de resultados y peticiones generadas por el usuario almacenados en las bases de datos. Para obtener esta información, hace uso de otros servicios tales como ResultService o LocationService.

  Antes de continuar, conviene explicar de qué manera se gestionan los datos en este proyecto basado en microservicios de Spring Boot. Primero, cuenta con objetos denominados **DTO** (Data Transfer Object, Objeto de Transferencia de Datos) y **DAO** (Data Access Object, Objeto de Acceso a Datos). [18]

  Los primeros simplemente se encargan de almacenar y entregar sus propios datos, reduciendo el costo de la comunicación entre servicios transfiriendo todos los datos necesarios en una sola llamada en lugar de varias. Por otro lado, los DAO son componentes de software que suministran una interfaz común entre la aplicación y los dispositivos de almacenamientos de base, como pueden ser las bases de datos.

  Todos los DTO del proyecto extienden del genérico BasicDTO, quien gestiona características globales de los mismos como el estado. Después, cada tipo de dato tiene su propio DTO.

  LocationDTO es la entidad encargada de gestionar las peticiones del usuario, almacenando sus datos: id, latitud, longitud, radio y fecha pedida. ResultDTO maneja

los resultados a través de su id, id del usuario, media obtenida, media basada en el momento de la semana, media en la franja horaria laboral y media en la franja horaria de descanso. RoleDTO cuenta con la información relativa a los posibles roles de usuario, para lo que es suficiente almacenar el nombre del rol y su id. Por último, UserDTO cuenta con el id del usuario, su nombre de usuario, contraseña, roles que posee y los respectivos id de sus roles.

Cada DTO cuenta con su respectivo DAO. Hay dos tipos de DAO en la aplicación, los que extienden del genérico de SQL y los que extienden del genérico de Mongo. Como los resultados de las peticiones son los únicos datos que se almacenan en la base de datos Mongo, el único DAO que extiende de BasicMongoDAO es ResultDAO, quién cuenta con una etiqueta @Document indicando la colección (Predictions) a la que pertenece. Este, así como el resto de DAO, tiene los mismos campos que su respectivo DTO.

Los DAO que extienden de BasicSQLDAO son LocationDAO, RoleDAO y UserDAO, datos que se almacenan en la base de datos SQL. Todos ellos están marcados con la etiqueta @Entity y @Table (indicando el nombre de la tabla a la que se asocia la entidad a la base de datos). Para que la relación entre usuarios y roles se establezca correctamente es necesario que se cree una tabla user_roles donde, a través de sus respectivos Ids, queda indicado que usuario tiene cada rol.

Como todas las tablas se crean automáticamente cuando la aplicación empieza a ejecutarse, es necesario indicar en UserDAO la relación que se quiere generar con los roles. Para ello, se utiliza la etiqueta @ManyToMany y @JoinTable en el set de roles que contiene el usuario.

Para comunicar la aplicación con las bases de datos se requiere utilizar repositorios, uno para cada DAO y DTO generado. Como es el caso de los DAO, se ha creado un repositorio genérico para las tablas SQL y otro para las colecciones de Mongo, cada uno con sus correspondientes etiquetas y extensiones. El único repositorio que extiende del genérico de Mongo es ResultRepository, ya que los resultados de las peticiones son los únicos que se almacenan en esta base de datos. Éste tiene un método para buscar resultados por id: findByUserId. RoleRepository tiene un método similar para buscar por un rol específico: findByRole. Los servicios hacen uso de estos repositorios para acceder a las bases de datos.

4.2.2 Servicios Amazon

Para el desarrollo del sistema se han utilizado dos servicios de Amazon Web Services: S3 y EMR (Elastic MapReduce). El primero se trata de la herramienta de almacenamiento de datos de Amazon, donde se encuentran los datos que el algoritmo utiliza para llegar a los resultados. Por otro lado, se encuentra también almacenado el algoritmo desarrollado en Scala con SparkSQL en forma de JAR que procesa el step del cluster.

Este último aspecto se implementa utilizando la segunda herramienta: EMR. Para procesar los datos se utilizan clusters configurados y levantados mediante código JAVA, que se explicará en la siguiente sección. Estos cluster, así como los datos de S3 a procesar se encuentran almacenados en los servidores de Amazon de Irlanda.

Más adelante se explicará el uso de otros servicios de Amazon como EC2 o ECS, utilizados para el despliegue de la aplicación de forma online.

4.2.3 Integración Amazon EMR-Plataforma

La integración de Amazon EMR y la plataforma web se hace mediante herramientas como colas RabbitMQ y la API de Amazon. Las tres clases Java del proyecto prediction-service esenciales para comprender la integración son: CustomMessageSender, QueueManager y PetitionManager.

La primera configura y habilita la utilización de RabbitMQ en el sistema, dándole nombre a las colas y estableciendo el @Component necesario. Se utilizan dos colas distintas de nombres prediction.finish y prediction.step, configuradas en la cuenta de RabbitMQ de la empresa Infinia Mobile, almacenada también en Amazon.

La primera cola, prediction.finish, la utiliza el endpoint de la API encargado de añadir la petición de localización que pide el usuario a la base de datos, ya que también envía un mensaje a través de esta cola al servicio QueueManager. Este mensaje contiene el id de la petición, para que el servicio pueda recuperarla y levantar el cluster correspondiente en Amazon EMR (si no hay uno levantado ya) y crear el step mandando los datos de latitud, longitud, radio, id de la petición, fecha e id del usuario que ha realizado la petición al JAR almacenado en Amazon S3. Si se desea visualizar de forma gráfica el funcionamiento de la aplicación, se recomienda acceder al apartado F de la sección anexos, donde se incluye un diagrama de secuencia.

Cuando el JAR termina de realizar los cálculos necesarios, utiliza la cola prediction.step para comunicarse con el servicio PetitionManager e informarle de que la petición ha terminado de procesarse para poder actualizar el estado de esta en la base de datos. A continuación, se presenta un diagrama de la aplicación actualizado tras lo explicado con más detalle:

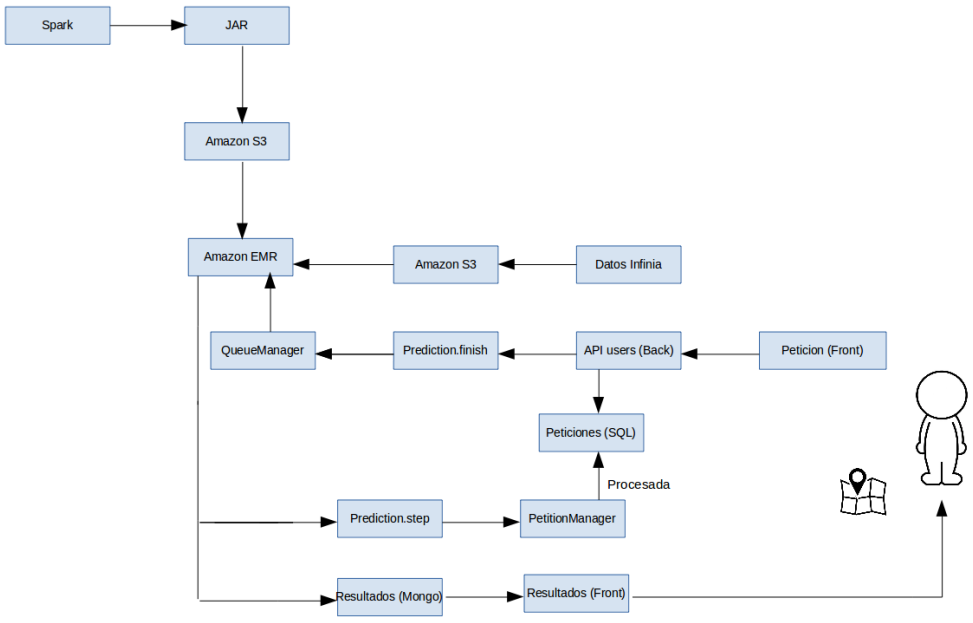


Figura Desarrollo-20: Esquema integración Amazon-Plataforma

4.2.4 API

El sistema cuenta con una API desarrollada para este Trabajo de Fin de Grado que se encarga de realizar todas las peticiones a la aplicación necesarias para su correcto funcionamiento. Sin necesidad de utilizar la aplicación web de forma gráfica, un usuario podría utilizar las funciones descritas a continuación para realizar sus peticiones y obtener resultados:

- /login

Es necesario autenticarse para poder realizar las peticiones al resto de funciones. Para ello es necesario enviar una petición POST con los datos: username y password.

- /prediction/users/petition

Encargado de añadir la petición a la base de datos y de comunicarse a través de RabbitMQ con QueueManager, quien procesa la petición en Amazon EMR. Para utilizar esta función es necesario pasarle a través de una petición POST los siguientes parámetros: latitud, longitud, radio y date (en formato año-mes-día).

- /prediction/users/resultados

Basta con enviarle una petición POST con el parámetro: userId indicando de qué usuario se desea obtener los resultados. Si el id de usuario corresponde a un administrador y el usuario ha hecho login como tal, se obtendrán los resultados de todos los usuarios. Por el contrario, si el usuario tiene como rol User y no Admin, se le devolverán solamente los resultados de sus peticiones.

- /prediction/users/peticiones

Funciona de la misma manera que la función de resultados. La única diferencia es que en lugar de mostrar los mismos, devuelve las peticiones realizadas.

- /prediction/users/registration

Esta función solamente puede ser accedida por los administradores. Para añadir un usuario a la base de datos y que posteriormente pueda hacer login en la aplicación, es necesario enviarle una petición POST a esta función con los parámetros: username, password y role, donde éste último puede ser USER o ADMIN en función de los privilegios que se le quiera otorgar al nuevo usuario.

- /prediction/users/activeRoleUser

Para hacer uso de esta funcionalidad basta con enviarle una petición GET. Devuelve true si el usuario que ha hecho login en la aplicación es un administrador y false si es un USER.

4.2.5 Detalles de implementación

Para gestionar las sesiones de usuario el sistema utiliza Redis, un motor de base de datos en memoria basado en el almacenamiento en tablas de hashes. Para ello se despliega una imagen de Redis en Amazon ECS, como del resto de servicios de la aplicación.

La clase SessionConfig de gateway-service habilita la utilización de Redis en la aplicación mediante la etiqueta @EnableRedisHttpSession.

Además de esta característica, cuando se crean por primera vez las tablas de la base de datos SQL (al levantarse prediction-service si no están creadas ya), se añade un usuario con nombre de usuario y contraseña "admin" y con rol ADMIN de forma que se puedan probar todas las características de la aplicación desde un principio.

4.2.6 Algoritmo de predicción y JAR

Para la realización del JAR se ha utilizado el lenguaje de programación Scala, el framework Apache Spark, Spark SQL y el IDE IntelliJ.

A la hora de realizar la predicción, el usuario introduce a través de la plataforma web como parámetros de entrada: latitud, longitud, radio y fecha. Tras pasar por el algoritmo, se obtiene un resultado de cuánta gente habrá en esa latitud y longitud, con ese radio y en esa fecha. El resultado se calcula mediante 4 aproximaciones, que serán explicadas a lo largo de este punto.

Como primer paso para la realización de las predicciones, es necesario leer los datos de las localizaciones que se encuentran almacenados en S3. Una vez leídos, se comprueba los parámetros de la petición que ha realizado el usuario para trabajar con ellos. A continuación, se filtran las localizaciones para trabajar únicamente con las que entren dentro del área pedida por el usuario.

Las tablas de datos con las que se trabaja cuentan con diversa información de la localización que almacena: token de la aplicación a través de la cual ha recibido la información, id del usuario, latitud, longitud y fecha en formato timestamp. Respecto estos datos, para el cálculo de la predicción es necesario añadir una columna que cuente con la fecha en formato año-mes-día y el día de la semana que corresponde a cada fecha.

Una vez se tienen todos los datos necesarios y en el formato deseado, se procede al cálculo de la primera aproximación del resultado mediante la media. Para ello, se seleccionan todos los usuarios distintos que estuvieron en las diferentes fechas con las que contamos, se agrupa la información por las mismas, se cuenta la cantidad de personas que hubo por cada una de ellas y se realiza la media de personas que estuvieron en esas fechas.

Tras haber obtenido la primera aproximación de predicción, se pasa a realizar una más precisa. Ésta tiene en cuenta el día de la semana para el que el usuario ha pedido el cálculo. Se filtra por día laboral (entre semana) y día de descanso (fin de semana), ya que normalmente existe una notable diferencia entre la cantidad de gente en localizaciones como centros comerciales o parques en estos dos periodos de tiempo. Para ello, filtramos

la información con la que se trabaja relativa al momento de la semana en nuestro set de datos y se trabaja con los datos resultantes de la manera mencionada anteriormente para calcular una nueva media.

Como tercera y cuarta aproximación, se trabaja con dos momentos distintos del día: horario laboral (6 am – 8pm) y horario de descanso (8pm – 6 am), donde se engloban la mayoría de los horarios laborales. Para realizar este cálculo, se añade una nueva columna a nuestro set de datos donde se indica si se trata de una localización obtenida durante la noche o durante el día. De la manera mencionada en el primer cálculo de una media simple, se vuelve a obtener la media, pero esta vez con datos más acotados y precisos, obteniendo una media de personas para cada uno de los horarios. Hay que tener en cuenta que es posible que un usuario haya sido registrado más de una vez durante el día, pudiendo ser en una ocasión durante horario laboral y otra durante horario de descanso, por lo que la cantidad de personas en ambas franjas no tiene porqué sumar la cantidad de usuarios distintos totales.

Una vez obtenidas las cuatro aproximaciones del resultado, se pasa a almacenar los resultados en la base de datos de MongoDB. Para ello se establece conexión con la misma, se crea un documento con las predicciones calculadas, así como con información de la petición solicitada (usuario, latitud, longitud y radio). Por último, se inserta en la base de datos indicando la colección y el cliente de MongoDB.

Cuando el resultado ha sido almacenado correctamente se pasa a notificar, mediante un mensaje en una cola de RabbitMQ, al back-end de la plataforma para que éste pueda mostrar la información correspondiente en el front-end y cambie el estado de la petición en base de datos de Pendiente a Finalizada.

Este algoritmo lo procesa un cluster de Amazon cuando lo levanta el back-end de la plataforma al recibir una petición. Para ello tiene que estar almacenado en un lugar accesible para los servidores de Amazon, como es en este caso en S3 en formato JAR. Para crear el archivo JAR con el algoritmo se ha utilizado una "assembly-task" que comprime los archivos Scala trabajados, tarea que ha sido automatizada con Jenkins.

4.2.7 Configuración de las máquinas de Amazon

Para que la plataforma funcione correctamente es necesario tener varias máquinas de Amazon trabajando, por un lado, las encargadas de poner el sistema online y por otro las encargadas de realizar los cálculos del algoritmo con las peticiones recibidas de los usuarios.

Las primeras se han configurado utilizando las herramientas de Amazon ECS (Elastic Container Service) como se muestra en la figura 21, S3 (Simple Storage Service) y EC2 (Elastic Compute Cloud). Se tratan de un cluster con una instancia de tipo m4.large que cuenta con 2 CPU virtuales y 8 Gigabytes de memoria. En este cluster están desplegados todos los servicios de la plataforma (prediction-service, gateway-service, front-service, discovery-service y el servidor de Redis). Además, se cuenta con un cluster con una instancia de tipo t2.micro con 1 CPU virtual y 1 Gigabyte de memoria, donde se encuentra desplegado el servidor SQL.

Para que estas máquinas puedan desplegar los servicios, es necesario subirlos en forma de imágenes Docker a los repositorios Amazon ECR (Elastic Container Repository) y crear una task correspondiente para cada uno de ellos con los parámetros necesarios como, por ejemplo, las conexiones entre ellos (registros al discovery, envío de datos…) o a las bases de datos. Para poder tener un seguimiento de lo ocurrido en los servicios desplegados, se han creado unos logs donde se reporta el funcionamiento y estado de estos. Por otro lado, para que los puertos necesarios de cada servicio estén abiertos, se ha creado y configurado un "security-group" que permite las conexiones necesarias.

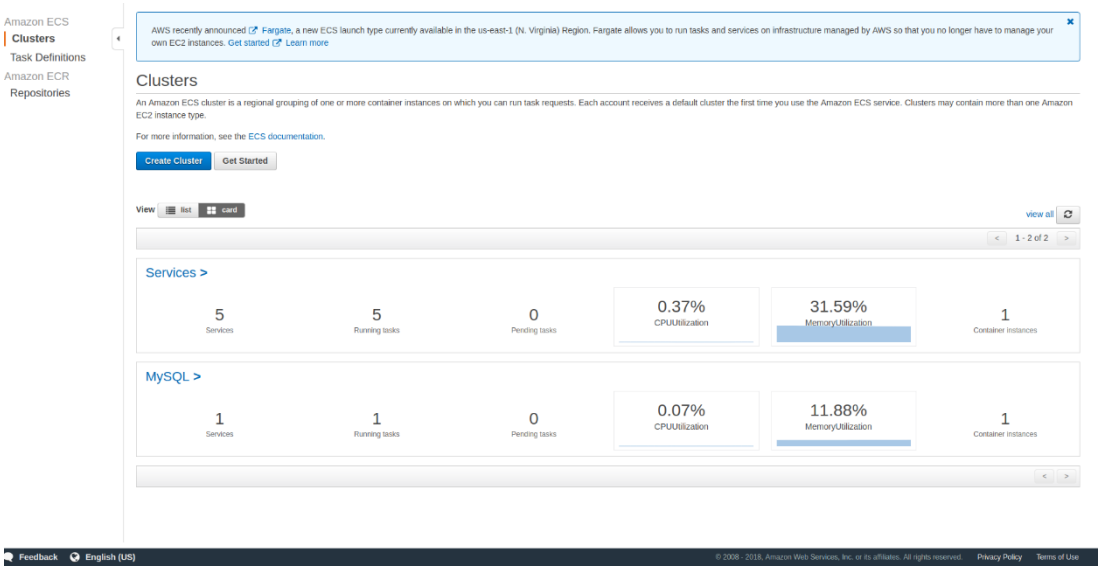


Figura Desarrollo-21: Clusters de Amazon ECS

Además de estas máquinas mencionadas, es necesario trabajar con la herramienta Amazon EMR (Elastic MapReduce) para llevar a cabo el procesamiento de las peticiones. Para ello se necesita mucha más capacidad de procesamiento que para desplegar los servicios, ya que se trabaja con mucha cantidad de datos.

Para procesar la información por el algoritmo se utilizan 13 máquinas, 12 Cores y un Master. Las 12 máquinas Core son del tipo r4.8xlarge, que cuentan con 32 CPU virtuales y 244 Gigabytes de memoria. La máquina Master es del tipo m4.4xlarge que cuenta con 16 CPU virtual y 64 Gigabytes memoria.

El back-end de la aplicación configura las máquinas cuando las levanta al recibir una petición, y éstas la procesan con el algoritmo de predicción almacenado en S3 en formato JAR.

4.2.8 Balanceador de carga para los servicios e IP elástica

Cuando llega una petición a los servicios desplegados en Amazon, es necesario configurar dos balanceadores de carga para que las peticiones se distribuyan correctamente. Por un lado, un balanceador de carga de tipo "red" tiene como función distribuir las peticiones entre los servicios Discovery, Prediction y Front. Por otro, uno de tipo aplicación gestiona las correspondientes al Gateway.

Cuando se hace un despliegue de las imágenes de Docker en los servidores de Amazon, éstos le asignan una IP diferente a cada servicio durante cada despliegue. Debido a esto, sería necesario cambiar la configuración de cada tarea que realizan los cluster (el de servicios y el de SQL) para adaptar las conexiones. Para evitar este tedioso proceso y poco automatizado, se ha creado una IP elástica y se le ha asignado al balanceador de carga de los servicios a través de Amazon EC2, como se puede observar en la figura 22.

Ahora las peticiones a los servicios se distribuyen de manera adecuada y su IP se mantiene aun cuando se reinicien. Esto último permite hacer despliegues sin tener que modificar datos de conexiones entre los servicios en cada ocasión.

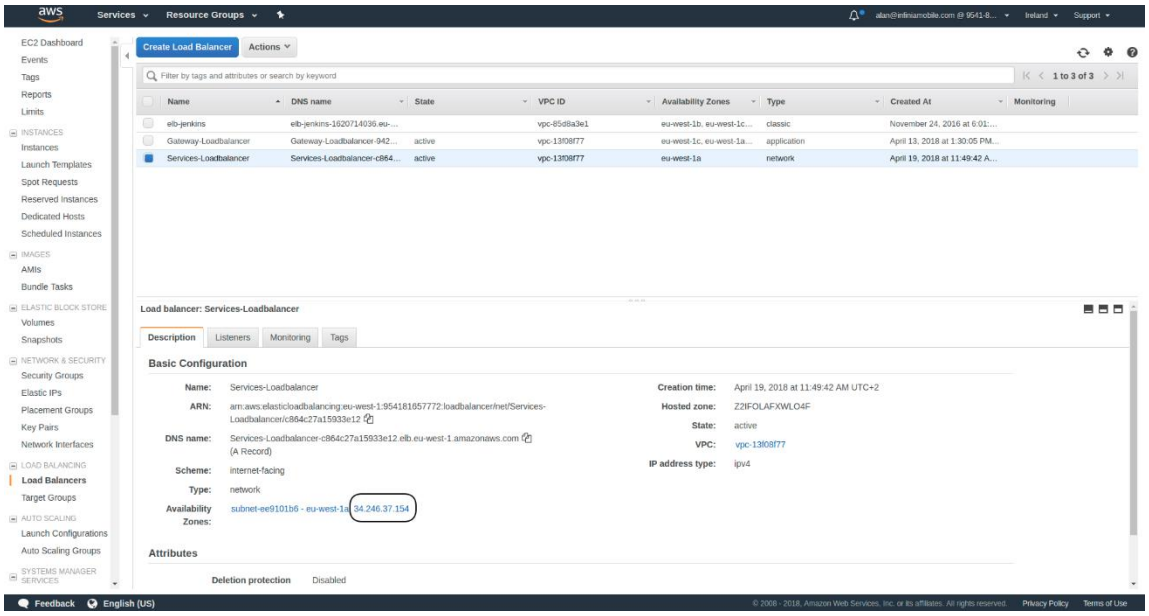


Figura Desarrollo-22: Balanceadores de carga de Amazon EC2

5 Integración, pruebas y resultados

La parte de integración ha sido explicada anteriormente en el desarrollo del proyecto, ya que era necesario para comprender su funcionamiento.

5.1 Pruebas API

Las pruebas a la API han sido desarrolladas con la herramienta PostMan y mediante llamadas a un entorno de desarrollo, que es el que se despliega en producción en Amazon para acceder mediante una URL no local. Para tener una representación gráfica de un caso de uso de la plataforma, se recomienda acceder al apartado D de los anexos donde se incluye un diagrama de caso de uso.

Para acceder a las funcionalidades de la plataforma es necesario hacer login tanto si se accede mediante la interfaz gráfica web realizada o mediante llamadas a la API. Para ello, se suministra al endpoint de la aplicación /login las credenciales username y password. El dominio localhost puede ser sustituido por la URL de producción que alberga la aplicación web en servidores Amazon. En la figura 23 se puede ver que se obtiene un mensaje de "Status: 200 OK", representando que el usuario ha hecho login correctamente mediante la petición POST:

Figura Integración, pruebas y resultados-23: Petición de login

Una vez hecho login en la aplicación, si el usuario desea realizar una petición a la plataforma, debe hacer una llamada al endpoint /prediction/users/petition con los datos de entrada pedidos: latitud, longitud, radio y fecha (date). En la figura 24 se muestra un ejemplo donde se devuelve el mismo código de estado, 200, que al realizar el login, indicando que la petición POST se ha llevado a cabo correctamente:

Figura Integración, pruebas y resultados-24: Petición de solicitud

Esto provoca que se levante un cluster, como se muestra en la figura 25, y se añada el step al mismo en Amazon EMR, como se muestra en la figura 26:

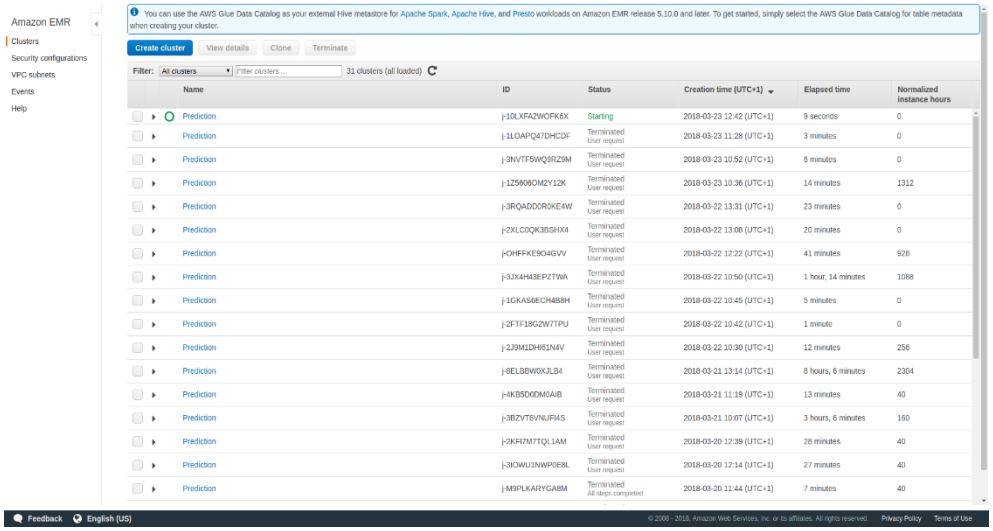


Figura Integración, pruebas y resultados-25: Cluster EMR iniciándose

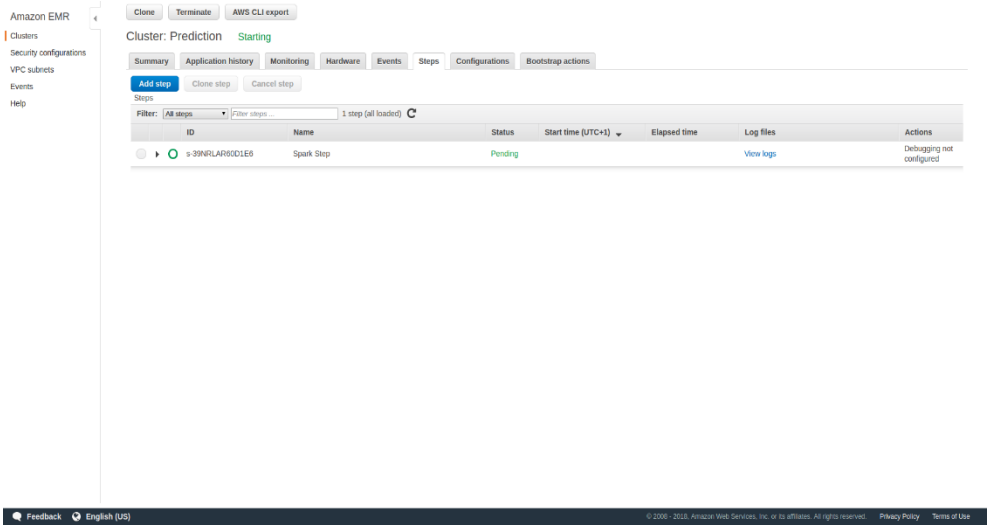


Figura Integración, pruebas y resultados-26: Step añadido al cluster EMR

Una vez se ha realizado una petición es posible visualizarla con su información asociada realizando una petición POST al endpoint prediction/users/peticiones. En el ejemplo de la figura 27, se puede apreciar las peticiones que, en ese momento, eran las que había en la base de datos con userId 2. Si no se indica id de un usuario específico, se visualizarían todas en caso de que el usuario autenticado sea administrador o, si el usuario autenticado tiene el rol "user", solamente visualizaría las suyas.

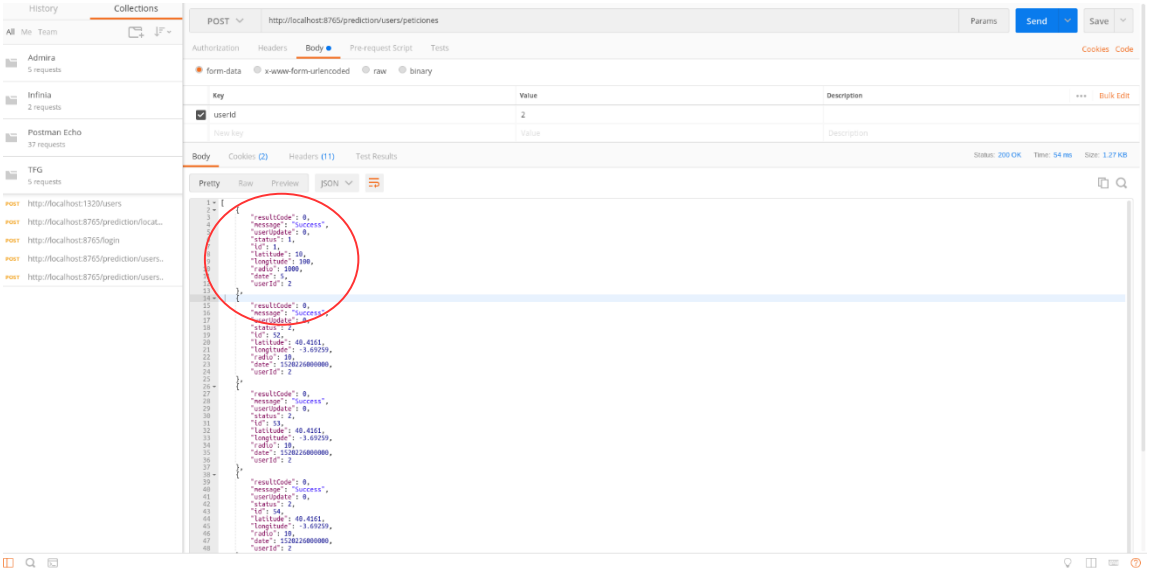


Figura Integración, pruebas y resultados-27: Petición de solicitudes

De la misma manera, si se desea ver los resultados obtenidos, al mandar una petición POST al endpoint prediction/users/resultados, se obtiene una respuesta con toda la información asociada a los resultados. En el ejemplo de la figura 28, se puede apreciar el resultado de una de las peticiones que, en ese momento, era la única en la base de datos que tenía el usuario con userId 2. Si no se indica id de un usuario específico, se visualizarían todos en caso de que el usuario autenticado sea administrador o, si el usuario autenticado tiene el rol "user", solamente visualizaría los suyos.

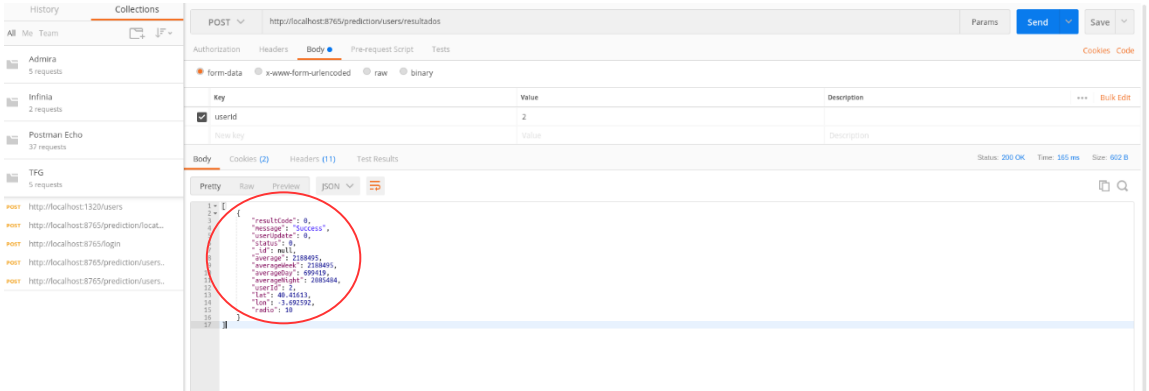


Figura Integración, pruebas y resultados-28: Petición de resultados

La aplicación cuenta con un endpoint que se utiliza para saber si el usuario que ha hecho login tiene rol de administrador o no. Para saberlo, basta con hacer una petición GET a prediction/users/activeRoleUser como se muestra en la figura 29, que devuelve true si el usuario es administrador o false si no lo es:

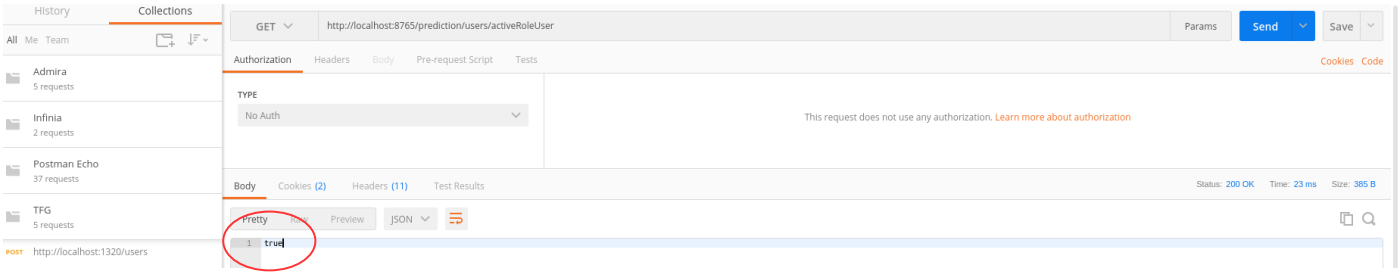


Figura Integración, pruebas y resultados-29: Petición a ActiveRoleUser

Por último, si se desea registrar un nuevo usuario, para lo cual es necesario haber hecho login con un usuario que tenga rol de administrador, basta con enviar una petición POST con los parámetros username, password y userRole (ADMIN o USER) que se desea que tenga el nuevo usuario registrado, como se muestra en el ejemplo de la figura 30:

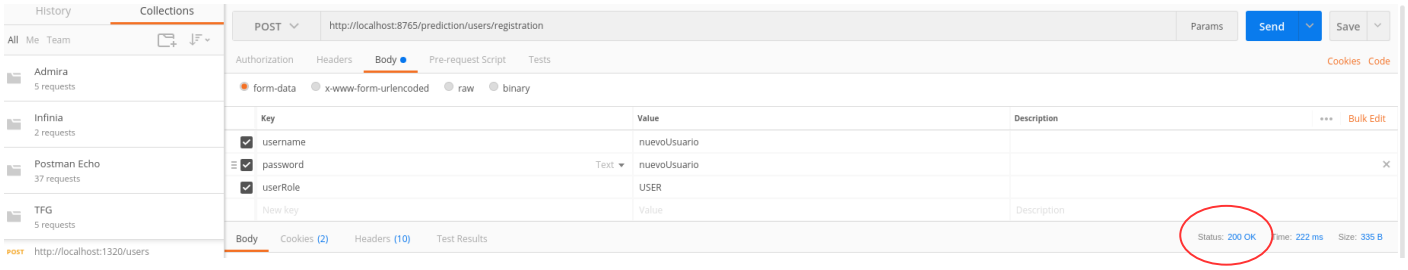


Figura Integración, pruebas y resultados-30: Petición de registro de usuario

5.2 Monitorización del cluster mediante Ganglia

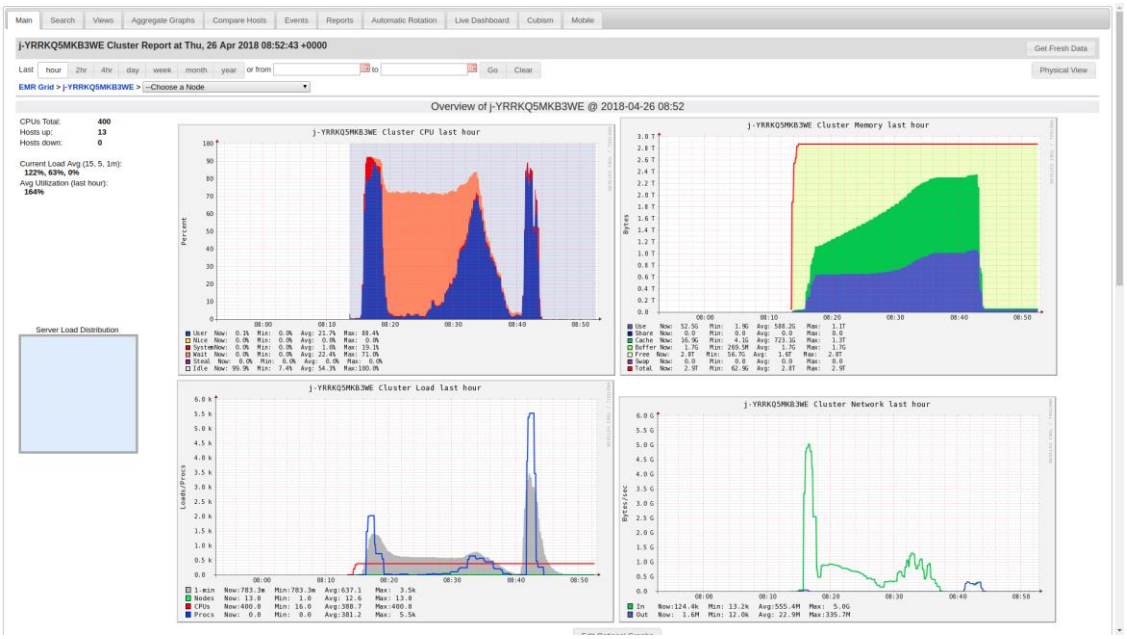


Figura Integración, pruebas y resultados-31: Visualización de la herramienta Ganglia

Cuando se levante un cluster para realizar las predicciones es posible añadirle una herramienta llamada Ganglia, utilizada para el monitoreo de forma distribuida y escalable para sistemas de cómputo de alto rendimiento, clusters y redes.

Como podemos observar en las gráficas, el tiempo que el cluster está trabajando utiliza casi la totalidad de sus CPU y memoria, evitando desperdiciar capacidad de cómputo.

5.3 Automatización mediante Jenkins

Para probar el correcto funcionamiento de la plataforma, se han desarrollado una serie de tests en el back-end del sistema. Éstos tests se pasan de forma automática, generando un reporte, cada vez que hay un cambio en el entorno de desarrollo almacenado en BitBucket. La aplicación cuenta con un total de 12 tests que se pasan en cada cambio, pudiendo visualizar en la siguiente figura que se realizan correctamente:

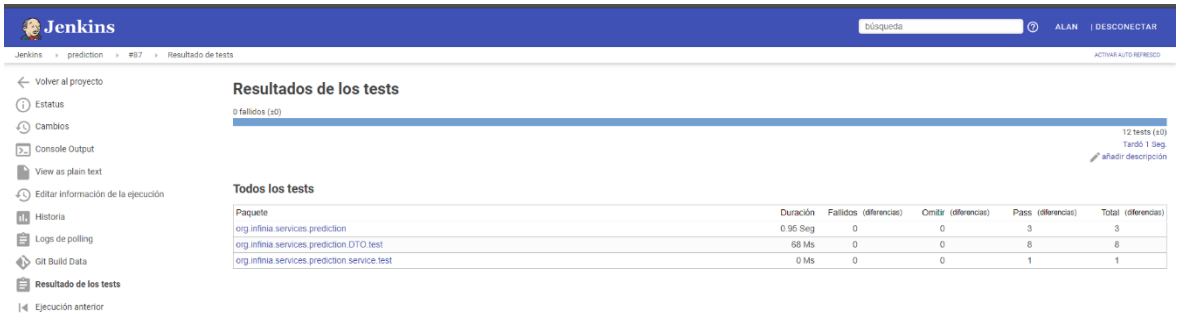


Figura Integración, pruebas y resultados-32: Test de la aplicación pasados correctamente

De esta manera, es fácil comprobar si los cambios realizados hacen que la aplicación siga funcionando correctamente o es necesario revertir alguno de ellos. Esto es posible gracias a la herramienta Jenkins, software de integración continua.

Otra de las opciones que suministra este software es la compresión de archivos en un Uber JAR, explicado anteriormente. Después de realizar los tests, Jenkins, cuyo panel de control se muestra en la figura 33, comprime los ficheros del proyecto de Scala que contiene el algoritmo y produce este JAR. A continuación, mediante una comunicación con los servidores de S3 de Amazon, sube el archivo para que la aplicación pueda trabajar con él para realizar las predicciones. De esta manera, basta con modificar el fichero donde se almacena el algoritmo desarrollado en Scala en el repositorio de Git para que el JAR producido sea directamente accesible desde la plataforma web.

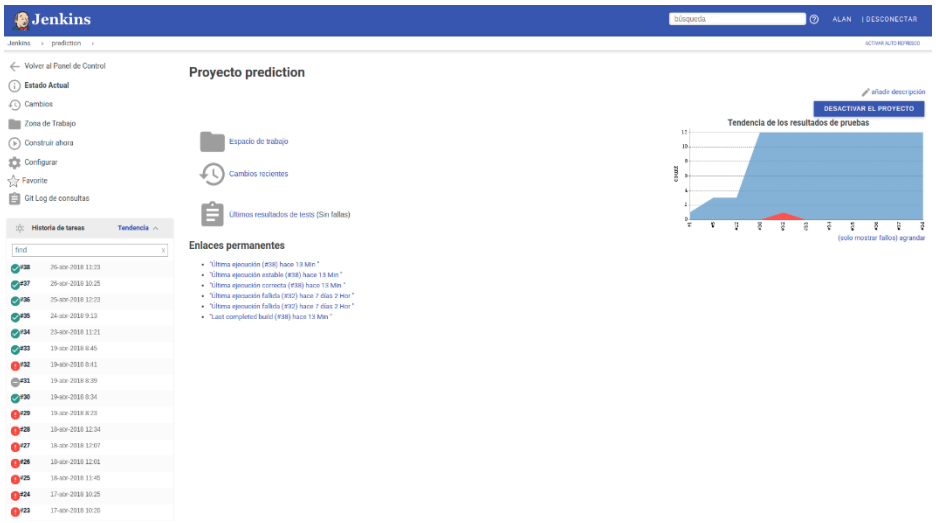


Figura Integración, pruebas y resultados-33: Panel de control de Jenkins

5.4 Resultados obtenidos

Al haber completado este Trabajo de Fin de Grado, obtenemos un producto final operativo. Como resultados del trabajo, podemos mencionar la plataforma desarrollada con todas sus componentes. Como ventaja principal, todos estos han sido desarrollados de forma modular, por lo que seguir trabajando sobre uno de ellos es tan fácil como hacerlo de forma aislada, ya que la integración es automática.

Por un lado está la plataforma web capaz de interactuar con el usuario de forma sencilla y práctica, dándole resultados a las peticiones que realicen de forma visual y gestionando el sistema de usuarios y de las sesiones de forma adecuada. Esta plataforma tiene desarrollada tanto la parte de front-end como la de back-end.

Por otro lado, la plataforma está desplegada en servicios web de Amazon, lo que permite que posea escalabilidad, adaptabilidad, accesibilidad y capacidad de integración con otras plataformas. Las peticiones de predicción también se resuelven en máquinas de Amazon, por lo que si en algún momento se trabaja con más datos, aumentar la capacidad de las mismas resulta sencillo.

Todos los servicios se comunican entre sí por sistemas como RabbitMQ o peticiones a la API, lo que permite abstraerse de cómo se está llevando a cabo el proceso, y centrarse en la tarea concreta que se quiere realizar. Además, a la hora de desplegar la aplicación,

las imágenes se realizan con Docker, por lo que resulta fácil y cómodo hacer cambios en el entorno de producción.

Además de lo mencionado, como característica que mejora en gran medida el trabajo de los desarrolladores para un trabajo futuro, todo el sistema pasa por un sistema de integración continua llamado Jenkins. Esto permite realizar tareas como modificar el algoritmo de predicción sin modificar nada más, y que automáticamente se integre con el resto de la plataforma. Jenkins compila y empaqueta el algoritmo junto con todas sus dependencias y lo sube al servidor S3 de Amazon al detectar un cambio en el repositorio de Git. También ejecuta los tests de forma automática para comprobar que los cambios no han supuesto ningún problema.

Por último, para mostrar la correcta implementación del proyecto realizado, se ha llevado a cabo la realización de un algoritmo que calcula las peticiones pedidas por los usuarios.

5.5 Eficiencia del algoritmo: train-test

Para realizar los cálculos de la petición se cuenta con datos de los anteriores 3 meses respecto del día actual. A modo de ejemplo de resultado, se han realizado 4 peticiones de prueba para el día 4 de mayo de 2018.

Para comprobar la validez de estos resultados se ha utilizado el método train-test [19]. Para ello se han hecho dos particiones de los datos del 50% cada una. La primera se ha utilizado para predecir el resultado y la segunda para ver realmente cuánta gente hubo el día pedido. Al realizar estos cálculos en la parte de train no se han utilizado los del 4 de mayo.

Al calcular el error de predicción se ha utilizado la siguiente fórmula:
Error = |Predicción – ValorReal|/ValorReal, ya que el valor de predicción podría ser superior o inferior al calculado y nuestro objetivo sería acercar el Error a 0. Se divide entre el valor real para obtener un error relativo. Los resultados obtenidos son:

Retiro – 0.7 km de radio:

La media general es: 30480 sobre 25361

Error de predicción: 0.2018

La media en función del día de la semana es: 29919 sobre 25361

Error de predicción: 0.1797

La media en horario diurno (6 am - 8 pm) es: 23135 sobre 18601

Error de predicción: 0.2437

La media en horario nocturno (8 pm - 6 am) es: 10695 sobre 8982

Error de predicción: 0.1908

Gran vía – 0.5 km de radio:

La media general es: 99428 sobre 121105

Error de predicción: 0.1789

La media en función del día de la semana es: 76194 sobre 121105

Error de predicción: 0.3708

La media en horario diurno (6 am - 8 pm) es: 75973 sobre 89686

Error de predicción: 0.1528

La media en horario nocturno (8 pm - 6 am) es: 37026 sobre 50241

Error de predicción: 0.2630

Plaza de Toros de Las Ventas – 0.2 km de radio:

La media general es: 10138 sobre 12099

Error de predicción: 0.1620

La media en función del día de la semana es: 8797 sobre 12099

Error de predicción: 0.2728

La media en horario diurno (6 am - 8 pm) es: 7853 sobre 9209

Error de predicción: 0.1472

La media en horario nocturno (8 pm - 6 am) es: 3961 sobre 4821

Error de predicción: 0.1781

Parque Juan Carlos I – 0.7 km de radio:

La media general es: 1749 sobre 2149

Error de predicción: 0.1861

La media en función del día de la semana es: 1367 sobre 2149

Error de predicción: 0.3636

La media en horario diurno (6 am - 8 pm) es: 1440 sobre 1691

Error de predicción: 0.1483

La media en horario nocturno (8 pm - 6 am) es: 457 sobre 602

Error de predicción: 0.2394

Analizando los resultados que hemos obtenido, podemos apreciar que la media en función del día de la semana mejora únicamente en el caso de El Retiro, en el resto de los ejemplos empeora. Esto puede ser porque el día 4 de mayo es un viernes y es considerado como día de la semana pero, en este caso, hubiera sido mejor considerarlo como fin de semana. Además, esa semana fue puente unos días antes, por lo que la afluencia de gente pudo haber variado.

Fijándose en el resultado de horario diurno y nocturno, en el caso de El Retiro se predice un número superior al real, mientras que en los demás es inferior. Específicamente, el resultado diurno mejora la predicción para todos los casos menos para éste. Esto puede ser debido a que el parque cuenta con una media superior al valor puntual de un día durante el horario lectivo.

Haciendo la media de cada aproximación, se concluye de los resultados que la media en horario diurno es la mejor aproximación de las cuatro. Estas pruebas son orientativas y a modo de ejemplo, como trabajo futuro se pueden realizar pruebas más exhaustivas comprobando los resultados en más localizaciones, días y épocas del año. También se puede comprobar en los días puntuales factores como el clima o localizaciones de eventos y ver como éstos afectan a los valores.

Podemos ver que el error ronda el 21 %, lo cual no está mal para una primera aproximación del algoritmo. Como se ha mencionado anteriormente, el algoritmo de predicción de datos es una prueba de concepto para el desarrollo de la plataforma, por lo que no es el objetivo principal del proyecto. Para mejorar el algoritmo como se ha explicado anteriormente y como se detallará en el apartado de trabajo futuro, sería tan fácil como modificar el fichero de código donde se almacena y la aplicación lo utilizaría directamente gracias al sistema de integración continua configurado y utilizado.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este Trabajo de Fin de Grado se ha desarrollado un sistema adaptable, escalable, modulado, integrable y con un fácil despliegue. Como prueba de concepto se ha implementado un sistema de predicción del número de personas en una determinada localización y fecha, lo cual permite al usuario interactuar con el sistema para mostrar sus funcionalidades y aportar a los clientes de Infinia Mobile información para sus campañas publicitarias.

En la sección Estado del Arte se evalúan tecnologías que se utilizan hoy en día para llevar a cabo proyectos similares y, como se puede observar, se ha utilizado la mayoría de éstas a lo largo del desarrollo del sistema. Desde el punto de vista de aprendizaje, este proyecto ha tenido un enorme valor, ya que al comienzo no se tenía conocimiento sobre el uso de las herramientas y tecnologías mencionadas a lo largo de todo el documento y, gracias a su uso práctico, se ha aprendido a manejarlas.

Por último, haciendo un análisis respecto a los requisitos planteados al principio de este proyecto, se puede comprobar que se han cumplido de forma satisfactoria e, incluso en algunos aspectos, se han superado ya que se ha optimizado o añadido funcionalidades a la plataforma.

6.2 Trabajo futuro

Tras haber realizado este proyecto para la empresa Infinia Mobile, la primera acción a realizar como trabajo futuro sería incorporar el Trabajo de Fin de grado a la plataforma de la empresa. Para ello, se incorporaría la parte de front-end, back-end, bases de datos, servicios de RabbitMQ y servicios de Amazon. Para los primeros dos aspectos, se incorporaría el código realizado adaptándolo como fuera necesario. Por ejemplo, el servicio de login se utilizaría el ya desarrollado por la empresa. En cuanto a apariencia, bastaría con incorporar alguna característica para adaptarla a la actual de la plataforma de Infinia Mobile, pero el cambio sería mínimo ya que se ha intentado utilizar el mismo formato en todo momento.

A la hora de integrar las bases de datos, el trabajo sería sencillo ya que simplemente habría que cambiar las conexiones y almacenamiento a las que la empresa utilice, y añadir las tablas y conexiones que se hayan creado. Por último, en cuanto a servicios de Amazon basta con migrar los servicios a la ubicación donde estén los de la empresa, para que todo funcione correctamente. El proyecto está pensado para que la integración sea sencilla y eficaz.

Por otro lado, en cuanto al algoritmo de predicción, se explorarían técnicas de aprendizaje automático y Deep Learning para desarrollar modelos de predicción más precisos. Además, también se puede incluir otra información a las predicciones utilizando APIs externas. Por un lado, existen proveedores de información meteorológica que pueden ser de utilidad a la hora de realizar la predicción. Por ejemplo, si durante un día pedido va a llover, se podría asumir que ese día habría más personas en un centro comercial que en un parque. Por otro lado, información acerca de los eventos próximos que tendrán lugar puede ser útil, ya que en localizaciones como teatros o estadios se llenarán en gran medida en días con eventos.

Otra opción es contemplar en qué tipo de lugar está pedida la predicción, si es un parque, museo, plaza, barrio residencial, etc.

Por último, en cuanto a mejoras del sistema para un trabajo futuro, si en un determinado punto se cuenta con una ampliación del número de datos utilizados para calcular las predicciones, se podría ampliar el número de máquinas con las que se trabaja en los servidores de Amazon para adecuarse a las necesidades.

Referencias

[1] Spring Boot. Fuente: Guías Spring Boot (https://spring.io/guides/gs/spring-boot/)

[2] Redis. Fuente: Página oficial Redis (https://redis.io/)

[3] Gradle. Fuente: Página oficial Gradle (https://gradle.org/)

[4] SparkSQL. Fuente: Página oficial SparkSQL (https://spark.apache.org/sql/)

[5] Amazon S3. Fuente: Página oficial S3 (https://aws.amazon.com/es/s3/)

[6] Amazon EC2. Fuente: Página oficial EC2 (https://aws.amazon.com/es/ec2/)

[7] Amazon ECS. Fuente: Página oficial ECS (https://aws.amazon.com/es/ecs/)

[8] Amazon EMR. Fuente: Página oficial EMR (https://aws.amazon.com/es/emr/)

[9] RabbitMQ. Página oficial RabbitMQ (https://www.rabbitmq.com/)

[10] Docker. Fuente: Página oficial Docker (https://www.docker.com/)

[11] Jenkins. Fuente: Página oficial Jenkins (https://jenkins.io/)

[12] IBM. Fuente: Página oficial de IBM (https://www.ibm.com/ibm/es/es/)

[13] SAP. Fuente: Página oficial de SAP (https://www.sap.com/spain/index.html)

[14] SAS. Fuente: Página oficial de SAS (https://www.sas.com/es_es/home.html)

[15] Oracle. Fuente: Página oficial de Oracle (https://www.oracle.com/es/index.html)

[16] Microsoft. Fuente: Página oficial de Microsoft (https://www.microsoft.com/es-es)

[17] POM. Fuente: Apache Maven Project "Introduction to the POM": (https://maven.apache.org/guides/introduction/introduction-to-the-pom.html)

[18] DTO DAO. Fuente: Blog tecnología (http://orgullo.users.sourceforge.net/blog/?p=177)

[19] Train Test. Fuente: Blog tecnología (https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7)

Glosario

API	Application Programming Interface
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
S3	Simple Storage Service
EMR	Elastic MapReduce
EPS	Escuela Politécnica Superior
UAM	Universidad Autónoma de Madrid
DTO	Data Transfer Object
DAO	Data Access Object
DB	Database
MQ	Message Queue
AJAX	Asynchronous JavaScript And XML
GUI	Graphical User Interface
MQ	Message Queue

Anexos

En esta sección de la memoria se explica mediante un manual de instalación cómo generar y desplegar las imágenes de Docker de nuestros servicios, así como se proporciona información adicional de la configuración de la aplicación mediante un manual del programador.

A. Manual de instalación

En este manual de instalación se procederá a explicar cómo realizar el despliegue de la aplicación desde local a los servidores de Amazon. Pongamos el escenario de que se han desarrollado cambios en la aplicación y que es el momento de juntar la rama de desarrollo con la rama master y realizar un despliegue a producción. Gracias a las tecnologías Docker y Amazon ECR el proceso es simple.

El primer paso es general en local las imágenes de Docker de los servicios que se han modificado. Pongamos que en este caso se trata del Prediction-service, por lo que nos desplazamos hacia donde tengamos el servicio y limpiamos las imágenes anteriormente generadas:

```
docker rm $(docker ps -a -q) -f
docker rmi $(docker images -q) -f
```

A continuación, ejecutamos la tarea de Docker buildDocker definida en build.gradle del servicio mediante el comando: ./gradlew build buildDocker. Una vez tengamos las imágenes que queremos desplegar correctamente generadas en local, pasamos a etiquetarlas para que sean más fácil de identificar una vez subidas al repositorio de ECR, por ejemplo:

```
docker tag “nombreDeLaImagen”:latest 954181657772.dkr.ecr.eu-
west- .amazonaws.com/“nombreDeLaImagen”:latest
```

Por último, solamente nos falta subir las imágenes al repositorio:

```
docker push 954181657772.dkr.ecr.eu-west-
1.amazonaws.com/“nombreDeLaImagen”:latest
```

B. Manual del programador

Una vez tenemos las imágenes de los servicios desplegadas en Amazon, ha hecho falta una configuración de EC2 y ECS para hacer posible el despliegue en la nube. En este manual del programador se explicará detalladamente los pasos que se han seguido para ello.

Como se ha explicado anteriormente, en Amazon ECS se han configurado dos Clusters, uno para los servicios y otro para el servidor MySQL. Para que estos cluster utilicen las imágenes subidas al repositorio, es necesario definir una task para cada servicio. En estas task es necesario indicar información respecto de los puertos donde ha de desplegarse, la conexión a la base de datos y sus credenciales, la dirección del discovery-service para hacer posible el registro y la ubicación donde debe generar los logs para ver el estado de la aplicación.

Una vez tenemos generadas estas task, le indicamos a los cluster que las ejecuten para seguir con el despliegue. Como se ha indicado en la sección de IP-Elástica, al haber asignado la misma al balanceador de carga del gateway, no se modifica por más que sea necesario reiniciar este servicio

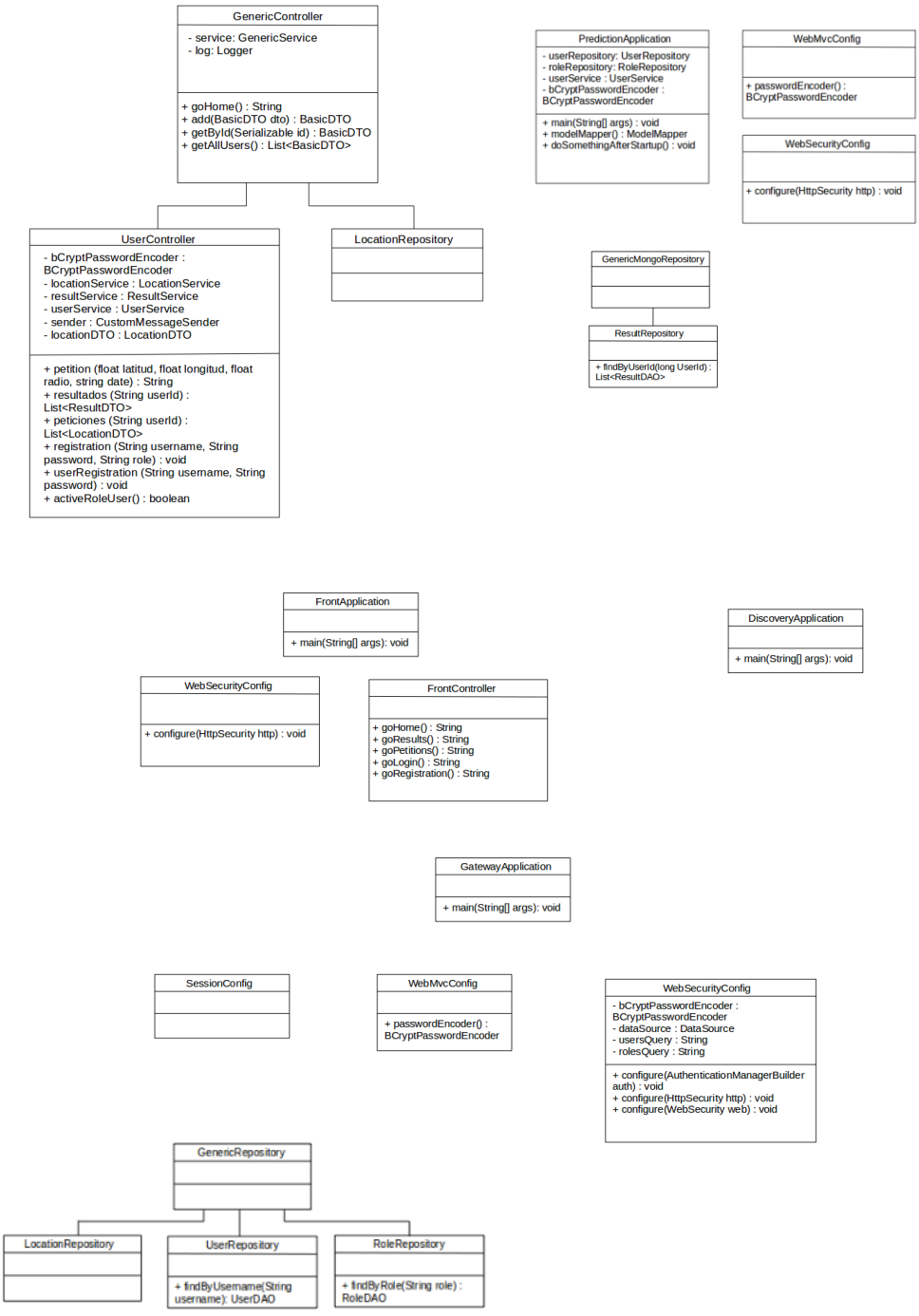
A nivel de programador, cuando un usuario realiza una petición es posible seguir el estado de esta a través de la consola de Amazon EMR. Si surge algún problema a la hora de procesar la petición, es posible ver los logs desde la consola, tanto “stdout” como “stderr”. Además, se puede acceder al “Resource Manager” del cluster para tener un seguimiento específico de las tareas que está realizando.

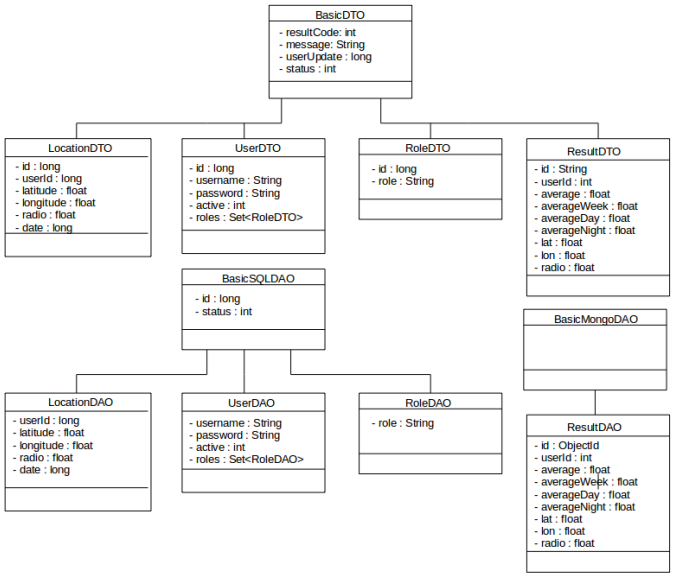
En cuanto a configuración EC2, es necesario configurar los dos balanceadores de carga, aplicación y red, y el security group que abre los puertos correspondientes a las conexiones de los distintos servicios. Una vez se tiene todo preparado, basta con levantar los servicios en los cluster de ECS para poner en marcha la aplicación.

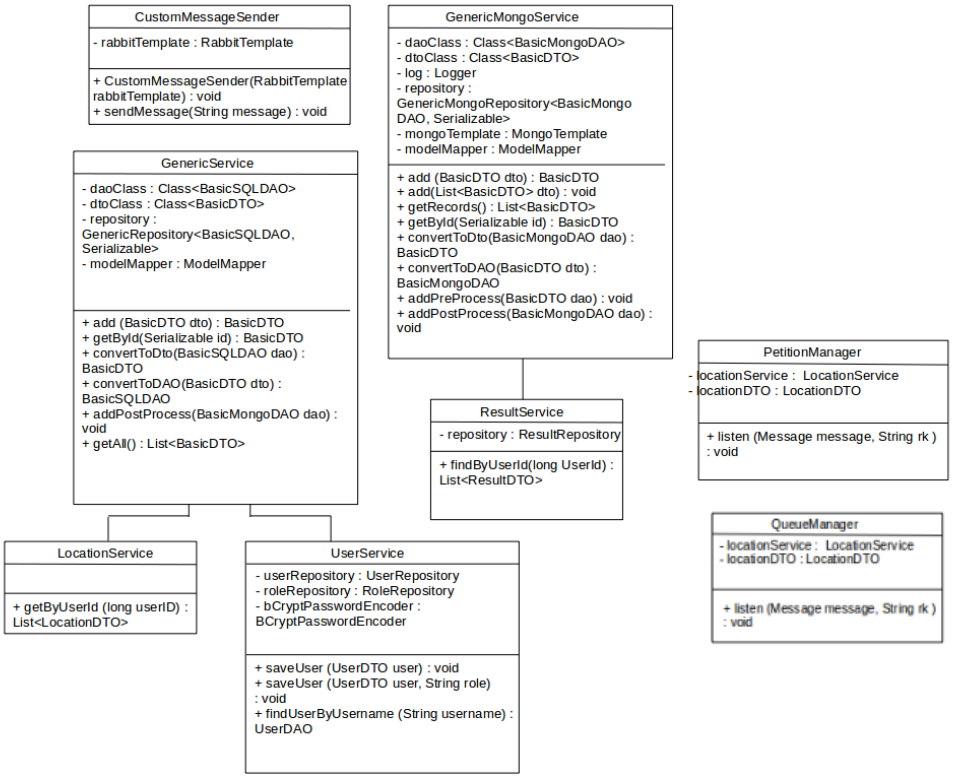
A continuación se muestran los diagramas desarrollados a lo largo del proyecto. Todos han sido referenciados durante la memoria, por lo que se indica en cada uno a qué lugar corresponden dentro de la lectura. Por razones de espacio han sido colocados al final, en la sección de anexos.

C. Diagrama de clases

Este diagrama ha sido referenciado en la sección 4.2.1

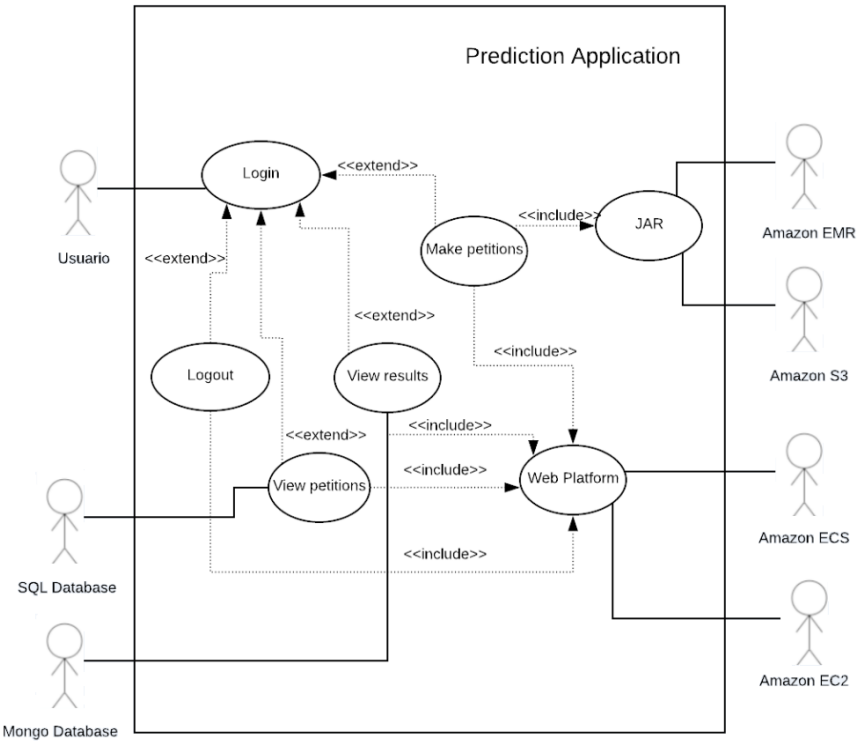



BasicDTO
- resultCode: int
- message: String
- userUpdate : long
- status : int
LocationDTO
- id : long
- userId : long
- latitude : float
- longitude : float
- radio : float
- date : long
UserDTO
- id : long
- username : String
- password : String
- active : int
- roles : Set<RoleDTO>
RoleDTO
- id : long
- role : String
ResultDTO
- id : String
- userId : int
- average : float
- averageWeek : float
- averageDay : float
- averageNight : float
- lat : float
- lon : float
- radio : float
BasicSQLDAO
- id : long
- status : int
BasicMongoDAO
LocationDAO
- userId : long
- latitude : float
- longitude : float
- radio : float
- date : long
UserDAO
- username : String
- password : String
- active : int
- roles : Set<RoleDAO>
RoleDAO
- role : String
ResultDAO
- id : ObjectId
- userId : int
- average : float
- averageWeek : float
- averageDay : float
- averageNight : float
- lat : float
- lon : float
- radio : float


CustomMessageSender
- rabbitTemplate : RabbitTemplate
+ CustomMessageSender(RabbitTemplate rabbitTemplate) : void
+ sendMessage(String message) : void
GenericMongoService
- daoClass : Class<BasicMongoDAO>
- dtoClass : Class<BasicDTO>
- log : Logger
- repository : GenericMongoRepository<BasicMongoDAO, Serializable>
- mongoTemplate : MongoTemplate
- modelMapper : ModelMapper
+ add (BasicDTO dto) : BasicDTO
+ add(List<BasicDTO> dto) : void
+ getRecords() : List<BasicDTO>
+ getById(Serializable id) : BasicDTO
+ convertToDto(BasicMongoDAO dao) : BasicDTO
+ convertToDAO(BasicDTO dto) : BasicMongoDAO
+ addPreProcess(BasicDTO dao) : void
+ addPostProcess(BasicMongoDAO dao) : void
GenericService
- daoClass : Class<BasicSQLDAO>
- dtoClass : Class<BasicDTO>
- repository : GenericRepository<BasicSQLDAO, Serializable>
- modelMapper : ModelMapper
+ add (BasicDTO dto) : BasicDTO
+ getById(Serializable id) : BasicDTO
+ convertToDto(BasicSQLDAO dao) : BasicDTO
+ convertToDAO(BasicDTO dto) : BasicSQLDAO
+ addPostProcess(BasicMongoDAO dao) : void
+ getAll() : List<BasicDTO>
ResultService
- repository : ResultRepository
+ findByUserId(long UserId) : List<ResultDTO>
PetitionManager
- locationService : LocationService
- locationDTO : LocationDTO
+ listen (Message message, String rk) : void
QueueManager
- locationService : LocationService
- locationDTO : LocationDTO
+ listen (Message message, String rk) : void
LocationService
+ getByUserId (long userID) : List<LocationDTO>
UserService
- userRepository : UserRepository
- roleRepository : RoleRepository
- bCryptPasswordEncoder : BCryptPasswordEncoder
+ saveUser (UserDTO user) : void
+ saveUser (UserDTO user, String role) : void
+ findUserByUsername (String username) : UserDAO

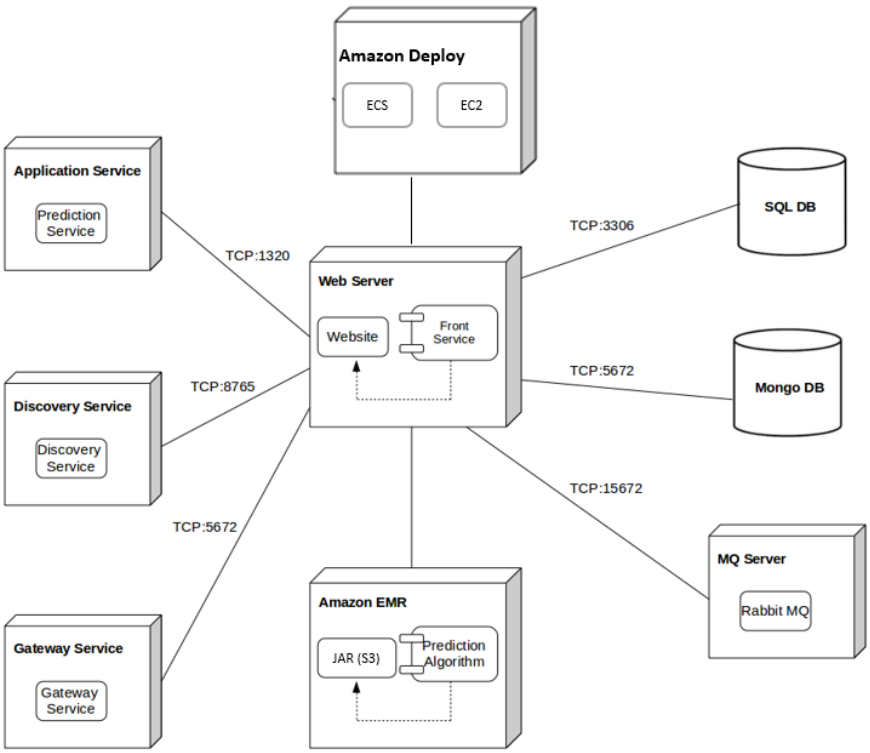
D. Diagrama de Caso de Uso

Este diagrama ha sido referenciado en la sección 5.1



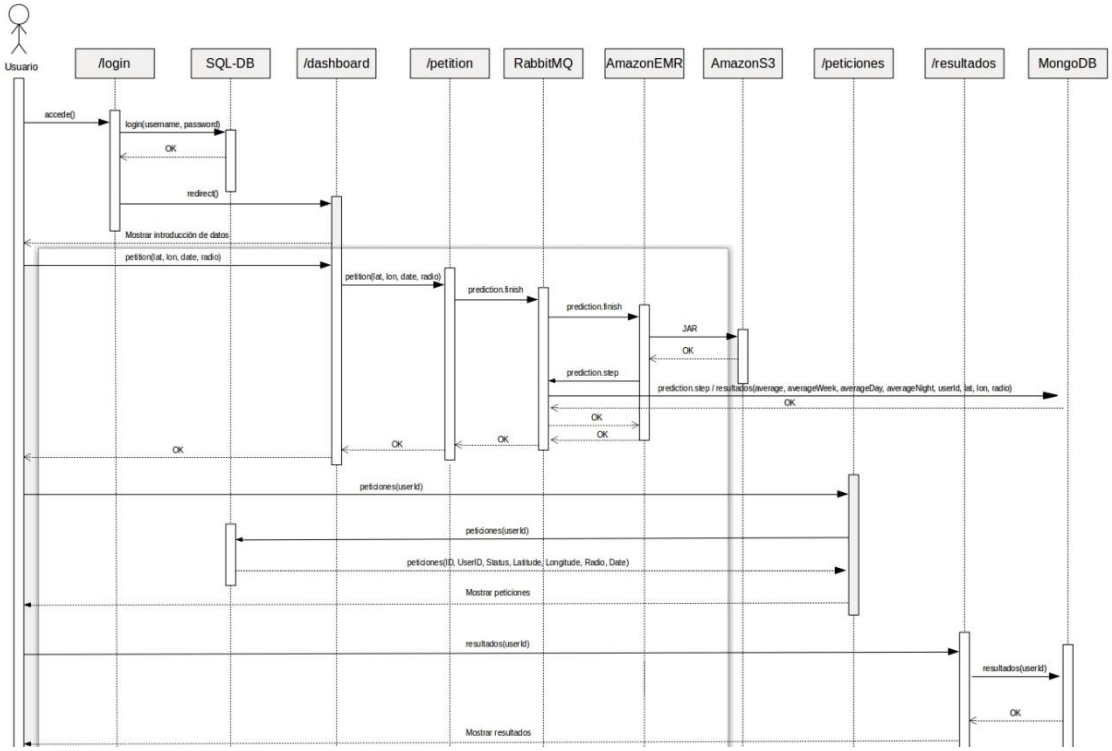
E. Diagrama de Despliegue

Este diagrama ha sido referenciado en la sección 2.2



F. Diagrama de Secuencia

Este diagrama ha sido referenciado en la sección 4.2.3



G. Ejemplo de utilización de la aplicación

A continuación se muestra un ejemplo de utilización de la plataforma para un usuario. Lo primero que tiene que hacer al acceder a la plataforma es proporcionar sus credenciales. En este caso, se proporciona "admin" como usuario y contraseña y se le da a login:

Figura Integración, pruebas y resultados-34: Login de la aplicación

Cuando el usuario entra en la aplicación, lo primero que se le muestra es la pantalla para realizar peticiones. Mediante el ratón se puede seleccionar el punto del mapa que se desea para hacer la petición, modificando el radio rellenando el campo. También se puede introducir manualmente la latitud y la longitud. En nuestro caso, se desea hacer una predicción para el Parque El Retiro en el día 4 de mayo de 2018 con un radio de 0.7 km. Tras seleccionar estos datos y pulsar el botón para enviar la petición, quedaría de la siguiente manera:

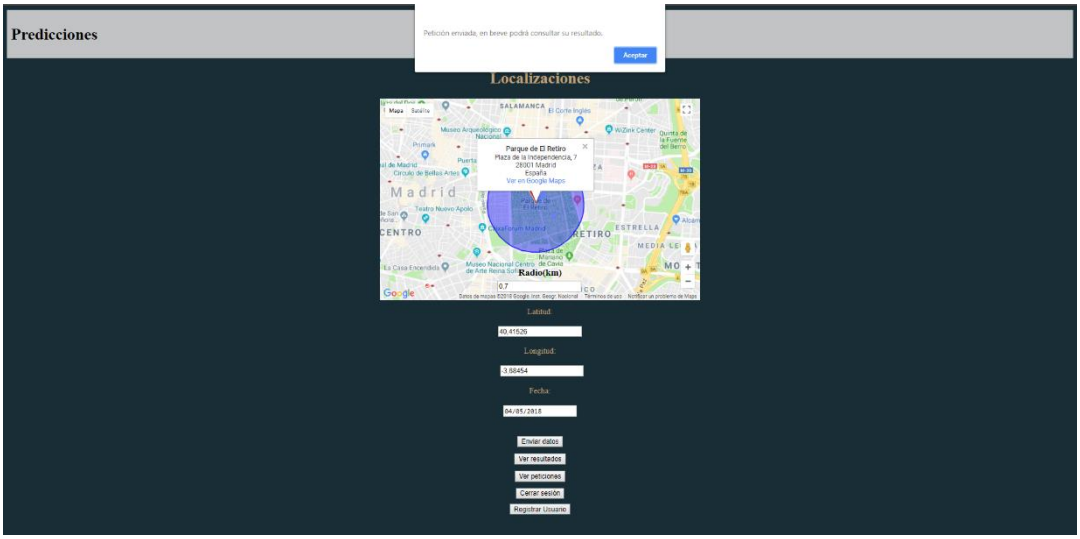


Figura Integración, pruebas y resultados-35: Petición de predicción realizada

El usuario no es consciente de todos los procesos que se realizan por detrás, ni la creación del cluster, ni la utilización del JAR con el algoritmo creado, ni las comunicaciones con las colas de mensajes RabbitMQ, etc. Si el usuario quiere acceder a sus peticiones, basta con seleccionar el botón "Ver peticiones", y estas se mostrarán en pantalla. En el mapa superior se puede visualizar la información de las peticiones pasando el ratón por encima, así como en la tabla inferior. El estado de la petición pasa de "Pendiente" a "Procesada" cuando se termina de procesar la petición.

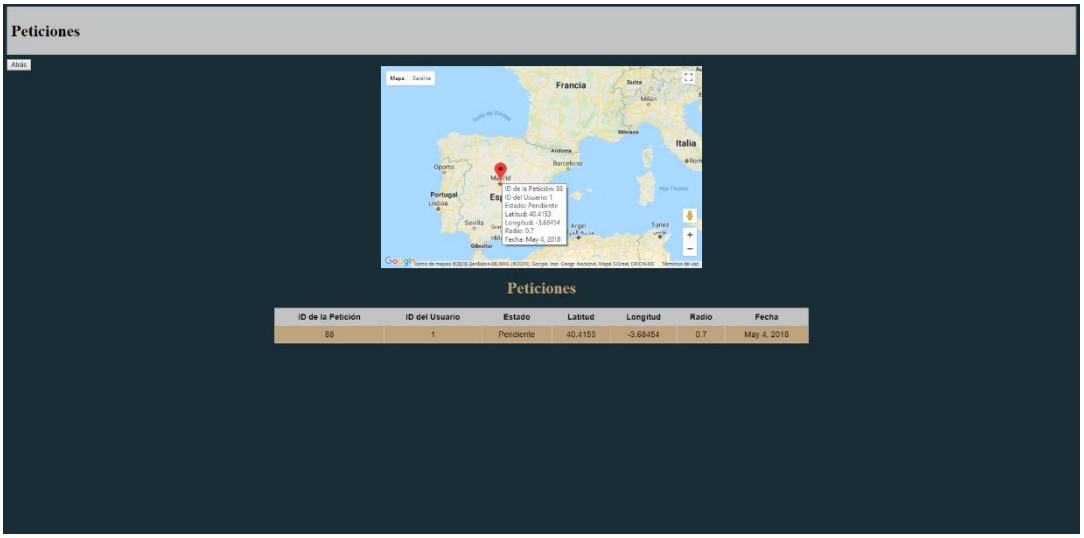


Figura Integración, pruebas y resultados-36: Visualización de peticiones

Cuando la petición se termina de procesar, el usuario puede ver su resultado volviendo atrás y seleccionando la opción "Ver resultados". Aquí se puede interactuar con el mapa de la misma manera, y también se visualizan los resultados en la tabla de información.

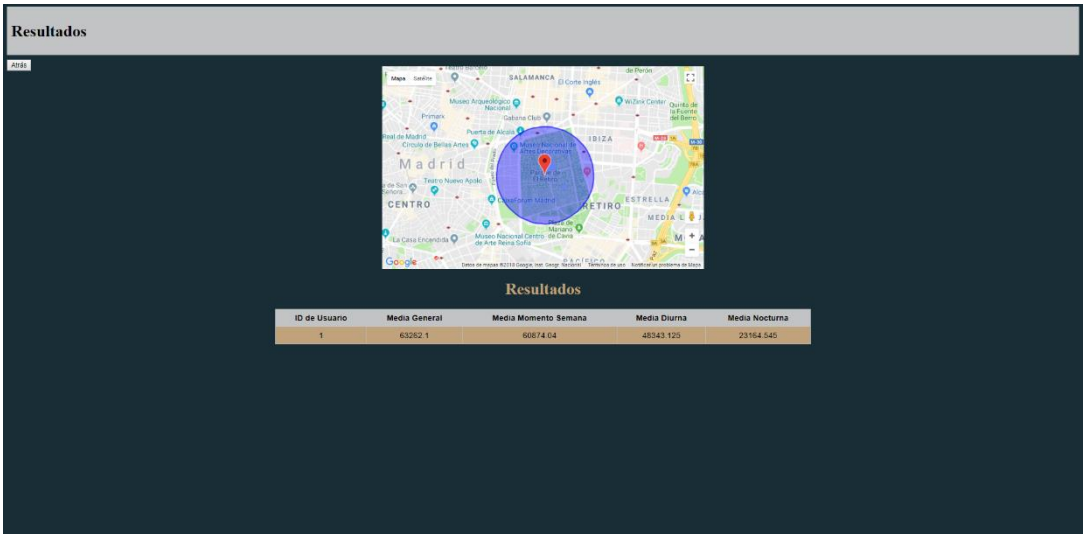


Figura Integración, pruebas y resultados-37: Visualización de resultados

Si el usuario que se ha autenticado es un administrador, éste puede registrar a nuevos usuarios. En el menú que se le muestra para ello tiene que rellenar el usuario y contraseña que desea para el nuevo usuario, así como su rol. Un usuario con rol de administrador puede ver todas las peticiones y resultados, mientras que un usuario con rol “User” (Usuario) solamente puede ver los suyos.

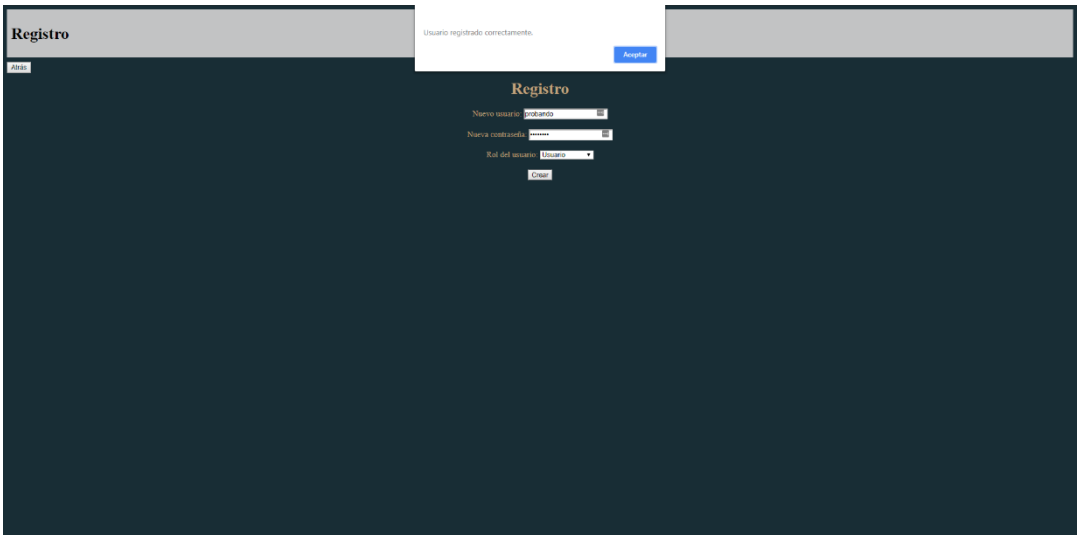


Figura Integración, pruebas y resultados-38: Registro de usuario

Por último, un usuario puede cerrar sesión en cualquier momento desde el menú principal de opciones, y volverá a la pantalla de login.